EnSight
Interface Manual

Table of Contents

0 Overview
1 User Defined Reader Version 1.0 API
2 User Defined Reader Version 2.0 API
3 User Defined Writer API
4 User Defined Math Functions
5 EnSight Command Driver
6 EnSight Python Interpreter

Computational Engineering International, Inc.
2166 N. Salem Street, Suite 101, Apex, NC 27523
USA • 919-363-0883 • 919-363-0833 FAX
http://www.ceisoftware.com

© Copyright 1994–2006, Computational Engineering International, Inc. All rights reserved.
Printed in the United States of America.

EN-IM Revision History

EN-IM:8.2-1	August 2006
EN-IM:10.0-1	January 2012

This document has been reviewed and approved in accordance with Computational Engineering International, Inc. Documentation Review and Approval Procedures.

Information in this document is subject to change without notice. This document contains proprietary information of Computational Engineering International, Inc. The contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, unless permitted by contract or by written permission of Computational Engineering International, Inc. Computational Engineering International, Inc. does not warranty the content or accuracy of any foreign translations of this document not made by itself. The Computational Engineering International, Inc. Software License Agreement and Contract for Support and Maintenance Service supersede and take precedence over any information in this document.

EnSight® is a registered trademark of Computational Engineering International, Inc. All registered trademarks used in this document remain the property of their respective owners.

CEI's World Wide Web addresses:
http://www.ceisoftware.com

Restricted Rights Legend

Use, duplication, or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Unpublished rights reserved under the Copyright Laws of the United States. Contractor/Manufacturer is Computational Engineering International, Inc., 2166 N. Salem Street, Suite 101, Apex, NC 27523 USA

Table of Contents

0 Overview

User Defined Reader APIs 0-1
How To Produce A User Defined Reader 0-3
User Defined Writers 0-7
User Defined Math Functions 0-7

1 User Defined Reader Version 1.0 API

1.1 Quick Index of Library Routines 1-2
1.2 Order Routines are Called 1-4
1.3 Detailed Specifications 1-6

USERD_bkup 1-7
USERD_get_block_coords_by_component 1-9
USERD_get_block_iblanking 1-10
USERD_get_block_scalar_values 1-11
USERD_get_block_vector_values_by_component 1-12
USERD_get_changing_geometry_status 1-14
USERD_get_constant_value 1-15
USERD_get_dataset_query_file_info 1-16
USERD_get_description_lines 1-17
USERD_get_element_connectivities_for_part 1-18
USERD_get_element_ids_for_part 1-20
USERD_get_element_label_status 1-21
USERD_get_extra_gui_defaults 1-22
USERD_get_extra_gui_numbers 1-23
USERD_get_global_coords 1-25
USERD_get_global_node_ids 1-27
USERD_get_name_of_reader 1-28
USERD_get_node_label_status 1-29
USERD_get_num_xy_queries 1-30

USERD_get_number_of_files_in_dataset 1-31
USERD_get_number_of_global_nodes 1-32
USERD_get_number_of_model_parts 1-33
USERD_get_number_of_time_steps 1-34
USERD_get_number_of_variables 1-35
USERD_get_part_build_info 1-36
USERD_get_reader_descrip 1-39
USERD_get_reader_release 1-40
USERD_get_scalar_values 1-41
USERD_get_solution_times 1-43
USERD_get_var_extract_gui_defaults 1-44
USERD_get_var_extract_gui_numbers 1-45
USERD_get_variable_info 1-47
USERD_get_variable_value_at_specific 1-48
USERD_get_vector_values 1-50
USERD_get_xy_query_data 1-52
USERD_get_xy_query_info 1-53
USERD_prefer_auto_distribute 1-54
USERD_set_extra_gui_data 1-55
USERD_set_filename_button_labels 1-56
USERD_set_filenames 1-57
USERD_set_time_step 1-58
USERD_set_var_extract_gui_data 1-59
USERD_stop_part_building 1-60

2 User Defined Reader Version 2.0 API

2.1 Quick Index of 2.0 Library Routines 2-2

2.2 Order Routines are Called 2-5

2.3 Routine History 2-9

At Version 2.00 2-12
At Version 2.01 2-12
At Version 2.03 2-13

At Version 2.04. 2-13
At Version 2.05. 2-13
At Version 2.06. 2-14
At Version 2.07. 2-14
At Version 2.08. 2-14
At Version 2.09. 2-15
At Version 2.10. 2-15
2.4 Detailed Specifications 2-16
USERD_bkup. 2-17
USERD_exit_routine 2-19
USERD_get_block_coords_by_component. 2-20
USERD_get_block_iblanking. 2-21
USERD_get_block_ghost_flags. 2-22
USERD_get_border_availability. 2-23
USERD_get_border_elements_by_type 2-24
USERD_get_changing_geometry_status 2-26
USERD_get_constant_val 2-27
USERD_get_dataset_query_file_info 2-28
USERD_get_descrip_lines. 2-29
USERD_get_element_label_status 2-30
USERD_get_extra_gui_defaults 2-31
USERD_get_extra_gui_numbers. 2-32
USERD_get_geom_timeset_number. 2-34
USERD_get_gold_part_build_info 2-35
USERD_get_gold_variable_info 2-41
USERD_get_ghosts_in_block_flag 2-43
USERD_get_ghosts_in_model_flag 2-44
USERD_get_matf_escalars_desc 2-45
USERD_get_matf_set_info 2-46
USERD_get_matf_set_type. 2-47
USERD_get_matf_var_info 2-48
USERD_get_matsp_info. 2-49
USERD_get_maxsize_info. 2-50

USERD_get_model_extents 2-52
USERD_get_name_of_reader 2-53
USERD_get_nfaced_conn 2-54
USERD_get_nfaced_conn_in_buffers 2-57
USERD_get_nfaced_nodes_per_face 2-62
USERD_get_node_label_status 2-65
USERD_get_nsided_conn 2-66
USERD_get_nsided_conn_in_buffers. 2-68
USERD_get_num_of_time_steps 2-72
USERD_get_num_xy_queries 2-73
USERD_get_number_of_files_in_dataset. 2-74
USERD_get_number_of_material_sets 2-75
USERD_get_number_of_materials 2-79
USERD_get_number_of_model_parts 2-80
USERD_get_number_of_species 2-81
USERD_get_number_of_timesets 2-82
USERD_get_number_of_variables 2-83
USERD_get_part_coords 2-84
USERD_get_part_coords_in_buffers 2-85
USERD_get_part_element_ids_by_type. 2-88
USERD_get_part_element_ids_by_type_in_buffers 2-90
USERD_get_part_elements_by_type 2-94
USERD_get_part_elements_by_type_in_buffers 2-96
USERD_get_part_node_ids 2-100
USERD_get_part_node_ids_in_buffers 2-101
USERD_get_periodic_ghosts_num_pairs. 2-104
USERD_get_periodic_ghosts_num_symmetry_faces 2-105
USERD_get_periodic_ghosts_pairs 2-106
USERD_get_periodic_ghosts_structured_face_info 2-108
USERD_get_reader_descrip. 2-112
USERD_get_reader_release. 2-113
USERD_get_reader_version. 2-114
USERD_get_sol_times 2-115
USERD_get_structured_reader_cinching 2-116

USERD_get_timeset_description 2-117
USERD_get_uns_failed_params 2-118
USERD_get_var_by_component 2-120
USERD_get_var_by_component_in_buffers 2-123
USERD_get_var_extract_gui_defaults 2-129
USERD_get_var_extract_gui_numbers 2-130
USERD_get_var_value_at_specific 2-132
USERD_get_vglyph_counts 2-134
USERD_get_vglyph_timeline_info 2-135
USERD_get_vglyph_timeline_times 2-136
USERD_get_vglyph_vector_info 2-137
USERD_get_vglyph_vector_values 2-138
USERD_get_vglyph_vector_xyzloc 2-139
USERD_get_xy_query_data 2-140
USERD_get_xy_query_info 2-141
USERD_load_matf_data 2-142
USERD_prefer_auto_distribute 2-144
USERD_rigidbody_existence 2-145
USERD_rigidbody_values 2-146
USERD_set_block_range_and_stride 2-148
USERD_set_extra_gui_data 2-150
USERD_set_filename_button_labels 2-151
USERD_set_filenames 2-152
USERD_set_right_side 2-153
USERD_set_server_number 2-154
USERD_set_time_set_and_step 2-155
USERD_set_var_extract_gui_data 2-156
USERD_size_matf_data 2-157
USERD_stop_part_building 2-159
USERD_use_periodic_ghosts 2-160

2.5 Converting a 1.0 API Reader to a 2.0 API READER 2-164

3 User Defined Writer API

What Information Can Be Provided By The API? 3-1
Example Writers 3-1
3.1 Directions For Writing Your Own UDW 3-3
Topical List Of User-Defined Writer API Methods 3-4
3.2 Routine Detail Specifications 3-7
USERD_writer_get_name 3-8
USERD_writer_get_writer_version 3-9
USERD_writer_write_geom 3-10

4 User Defined Math Functions

How the routines are invoked 4-1
Current Limitation 4-1
4.1 Detailed Routine Specifications 4-2
USERD_get_name_of_mf 4-2
USERD_get_mf_version 4-3
USERD_get_nargs 4-4
USERD_get_meta_data 4-5
USERD_evaluate 4-6
4.2 Example 4-7

5 EnSight Command Driver

Overview 5-1
5.1 Query Capability 5-4
Alphabetical List of Query Keywords: 5-4
Query Keyword Details 5-5
5.2 Routine Descriptions 5-33
enscmddriver_connect 5-33
enscmddriver_sendmesg 5-34
enscmddriver_query 5-36

enscmddriver_disconnect 5-37

6 EnSight Python Interpreter

Overview 6-1
General Python use in EnSight 6-3
Python and the EnSight command line 6-3
Limitations of the EnSight Python Interface 6-4

6.1 Python EnSight module interface 6-5

Interpreter startup 6-5
Ensight Module Methods 6-6
The Python native command language interface 6-22
Conversion of command language scripts to the native interface 6-23
Usage notes on the native Python interface 6-24
EnSight Python events 6-26
EnSight Python Code Methods 6-27

6.2 EnSight object API 6-31

6.3 Python EnVe module interface 6-32

The EnVe Movie object 6-32
The EnVe Image object 6-36
Additional EnVe API 6-42

6.4 EnSight extension mechanism 6-43

Internationalization and EnSight Extensions 6-48

6.5 Helper modules 6-49

0 Overview

EnSight has user defined capability for data readers, data writers, and math functions. This capability allows users to read and write data in custom ways, such as the handling of in-house data formats. There also exists limited capability to produce custom math functions for the variable calculator.

User Defined Reader APIs

The user defined reader capability included in EnSight can allow otherwise unsupported structured or unstructured data to be read. The user defined reader capability utilizes dynamic shared libraries containing routines defined in this document but customized by you, the user, (or some third party). This capability is available for all our supported architectures.

Two versions of this API are available.

API 1.0	Starting with EnSight Version 6.0, the 1.0 API was made available. It was designed to be friendly to those producing it, but requires more manipulation internally in EnSight. It may be a little easier to produce readers using this format (especially if a global coordinate array is a hallmark of your data format), but it requires more memory and processing time. It also has been frozen in capability - so it does not contain many of the newer features. **Underlying Philosophy** API 1.0 deals with: -> global coordinate array and corresponding -> global node id array -> global nodal variables -> for each part: -> local element connectivities (grouped by type) & corresponding -> local element ids -> local elemental variables The element connectivities, within parts, reference the global coordinate array. If node ids are provided, the element connectivities have to be in terms of the node ids. If node ids are not provided, the connectivities are in terms of the (one-based) index number of each node in the global coordinate array. Thus, node ids are more than labels - they are a part of the connectivity referencing scheme. Element ids are purely labels. This API was originally setup to try to make the interface to other codes as straightforward as possible. Efficiency was not the major consideration. EnSight must do a fair amount of work to get data provided in the manner described above into the form that it uses internally. There is mapping that has to be setup and maintained between the global arrays and the local part arrays so that updating over time can be accomplished efficiently. There is hashing that is required in order to deal efficiently with node ids. All of this leads to a considerable amount of temporary memory and processing, in order to get a model read into EnSight.

API 2.0	The current 2.0 API is considerably more efficient, and was designed more with that in mind. It lends itself closely to the EnSight Gold format. **Underlying Philosophy** API 2.0 deals with: -> for each part: -> part coordinates & corresponding -> part node ids -> part nodal variables -> part element connectivities (grouped by type) & corresponding -> part element ids -> part elemental variables API 2.0 requires that the coordinates and corresponding nodal variables be provided per part. This eliminates the global to local mapping with all its associated temporary memory and processing time. The connectivity of the elements in each part reference the node indices of its own (one-based) part coordinate array. The connectivity of the elements do not reference the nodes according to node ids. Node ids (and element ids) are purely labels for screen display and for query operations within EnSight. This eliminates the need for node id hashing as a model is read. The 2.0 API has been created for those needing more efficiency - both in terms of memory use and speed. The increased efficiency is possible because data is requested in a manner which more closely represents the way that EnSight stores and manipulates information internally. The new API requests size information and allocates the actual internal structures and arrays accordingly. Pointers to these arrays are passed directly to you in the routines which gather data, thus eliminating a considerable amount of temporary memory (and allocation time) that is needed in the 1.0 API. Depending on what you must do to get your data into the form required, the memory savings and the speed improvement when loading models can be quite significant! **Thus, some of its advantages and new features are**: * Less memory, more efficient, and faster - as indicated above. * Model extents can be provided directly, such that EnSight need not read all the coordinate data at load time. * Tensor and complex variables are supported * Exit routine provided, for cleanup operations at close of EnSight. * Geometry and variables can be provided on different time lines (timesets). * If your data format already provides boundary shell information, you can use it instead of the “border” representation that EnSight would compute. * Ghost cells (for both structured and unstructured data) are supported * User specified node and/or element ids for structured parts are supported * Material handling is supported * Nsided and Nfaced elements are supported * Structured ranges can be specified * Failed elements is supported * Material Species is supported * Rigid Body values can be supplied from the reader. * Reader can be allowed to deal with block min, max, and stride within itself - instead of having EnSight deal with it.

How To Produce A User Defined Reader

1. Write code for all pertinent routines in the library (Unless someone else has done this for you).

   This is of course where the work is done by the user. The word “pertinent” is used because depending on the nature of the data, some of the routines in the library may be dummy or optional routines.

   The source code for a dummy_gold library and for various other working or sample libraries is copied from the installation CD during installation. These will be located in directories under:

   ```
   $CEI_HOME/ensight100/src/readers
   ```

   Note: The directory following CEI_HOME in the path could vary depending on the version of EnSight installed.

   Examples:

   Basic dummy_gold routines provide skeleton for a new reader

   ```
   $CEI_HOME/ensight100/src/readers/dummy_gold
   ```

   Sample library which reads unstructured binary EnSight Gold data (version 2.08 API)

   ```
   $CEI_HOME/ensight100/src/readers/ensight_gold
   ```

   Sample library which reads C binary, 3D, static Plot3D data (version 1.0 API)

   ```
   $CEI_HOME/ensight100/src/readers/plot3d
   ```

   You may find it useful to place the source code for the library you create in this readers area as well, but you are not limited to this location.

   The descriptions of each library routine contained in version 1.0 API Detailed Specifications, and version 2.0 API Detailed Specifications, along with version 1.0 API Order Routines are Called, and version 2.0 API Order Routines are Called, as well other helps provided in this document, along with the example libraries, should make it possible for you to produce code for your own data reader.

2. Produce the dynamic shared library.

   This is a compiling and loading process which varies according to the type of machine you are on. In the user-defined-reader source tree we have tried to isolate the machine dependent parts of the build process using a set of files in the ‘config’ directory. In this directory there is a configuration file for each platform on which EnSight is supported. Before you can compile the installed readers you should run the script called ‘init’ in the config directory.

   i.e. (for UNIX)

   ```
   cd config
   ./init sgi_6.5_n64
   cd ..
   make
   ```

   If you are compiling for Windows, there are two options. If you have the Cygwin GNU utilities installed, you can use GNU make as for Unix. Otherwise, there is a script called makeall.cmd which will build all of the readers using nmake. The Makefiles in each reader directory will work using either make or nmake.

i.e. (WIN32 Cygwin)	**(using nmake)**

cd config sh init win32 cd .. make	cd config cp win32 config cd .. mkdir lib makeall.cmd

If you have platform-specific portions of code in your reader, the build system defines a set of flags which can be used within `#ifdef ... #endif` regions in your source, as shown in the table below.

Because the readers are now dynamically opened by EnSight, you may have to include dependent libraries on your link-line to avoid having unresolved symbols. If you are having problems with a reader, start ensight as "`ensight100 -readerdbg`" and you will get feedback on any problems encountered in loading a reader. If there are unresolved symbols, you need to find the library which contains the missing symbols and link it into your reader by adding it to the example link commands below.

If you choose to use a different build environment for your reader, you should take care to use compatible compilation flags to ensure compatibility with the EnSight executables, most notably on the SGI and HP-UX 11.0 platforms, which should use the following flags:

Machine Type	OS Flag	Shared Library Name Produced	LD Command used in Makefile
linux	-DLINUX	libuserd-X.so	ld -shared -o libuserd-X.so libuserd-X.o -lc
mac	-DMAC	libuserd-X.bundle	ld -bundle -twolevel_namespace -multiply_defined -suppress -undefined dynamic_lookup -o libuserd-X.so libuserd-X.o
ibm	-DIBM	libuserd-X.so	ld -G -o libuserd-X.so libuserd-X.o -bnoentry -becpall -lc

Once you have created your library, you should place it in a directory of your choice or in the standard reader location:

```
$CEI_HOME/ensight100/machines/$CEI_ARCH/lib_readers
```

For example, if you created a reader for "mydata", you should create the reader libuserd-mydata.so and place the file in your own reader directory (see section 3 below) or in the standard location:

```
$CEI_HOME/ensight100/machines/$CEI_ARCH/lib_readers/libuserd-mydata.so
```

3. By default EnSight will load all readers found in the directory:

```
$CEI_HOME/ensight100/machines/$CEI_ARCH/lib_readers
```

Files with names "libuserd-X.so" (where X is a name unique to the reader) are assumed to be user-defined readers.

There are two methods which can be used to supplement the default behavior.

(1) A feature which is useful for site-level or user-level configuration is the optional environment variable $ENSIGHT9_READER. This variable directs EnSight to load all readers in the specified reader directory (you should probably specify a full path) before loading the built-in readers. If the same reader exists in both directories (as determined by the name returned by USERD_get_name_of_reader(), NOT by the filename), the locally configured reader will take precedence.

(2) A useful feature for end-users is the use of the libuserd-devel reader. EnSight will search for a reader named libuserd-devel.so (.sl for HP or .dll for Windows). This reader can exist anywhere in the library path (see below) of the user. This is useful for an individual actively developing a reader because the existence of a libuserd-devel library will take precedence over any other library which returns the same name from USERD_get_name_of_reader().

As an example, a site may install commonly used readers in a common location, and users can set the ENSIGHT9_READER variable to access them:

```
setenv ENSIGHT9_READER /usr/local/lib/e8readers
```

A user working on a new reader may compile the reader and place it in a directory specified by the library path:

```
cp libuserd-myreader.so ~/lib/libuserd-devel.so
setenv <librarypath> ~/lib:$<librarypath>
```

The user is responsible for correctly configuring the library path variable in order to make use of the libuserd-devel feature. The library environment variables used are:

Machine Type	Environment Variable to Set
linux	LD_LIBRARY_PATH
mac	DYLD_LIBRARY_PATH
ibm	SHLIB_PATH

As always, EnSight support is available if you need it.

4. Use the **udr_checker** tool for help in debugging your user-defined reader.

The udr_checker.c routine can be used to debug EnSight User-defined readers. It exists because of the difficulty of debugging dynamic shared libraries when you don't have the source for the calling program (EnSight).

If udr_checker.c is compiled and linked with your reader source code (including access to any libraries needed, and the global_extern.h file), it will exercise most options of you reader, giving feedback as it goes. The resulting executable can be debugged using your favorite debugger. And if you have memory/bounds checking software (such as purify), you can (and should) run it with this executable to make sure that you are not overwriting things. ***Readers that bash memory will cause problems when run with EnSight!***

You will note that the Makefile provided with the readers in the EnSight distribution have a "checker" object. If you do a "make checker" instead of just a "make", the "checker" executable will be produced. You may need to modify these makefiles slightly if the locations of your reader files are different than the normal.

Once the "checker" executable exists, you can run the checker program by simply invoking it:

> checker

And you will be prompted for the type of information that you provide in the EnSight Data Reader dialog, namely:

The path
filename_1
[filename_2] Only if your reader uses two fields
swapbytes flag
<toggle flags> Only if your reader implements extra GUI

<pulldown flags> one flag value per line
<field contents> one field string per line

There are certain command line options that you can use to control some aspects of the checker program. One of the more useful is the ability to provide the input just described in a file. This is done in this fashion:

> checker -p <playfile>

And <playfile> would be a simple ascii file with 3 [0r 4] lines:

line 1: the path
line 2: filename_1
line 3: [filename_2] (if two_fields is TRUE)
line 3 or 4: 0 or 1, for swapbytes (0 is FALSE, 1 is TRUE)
remaining lines 0 or 1 for toggle disable enabled
one line for each toggle
0 - num_pulldown_values for pulldown choice
one line for each pulldown
strings
one line for each field

example playfile for an EnSight Gold reader casefile (entitled cube.play) could look something like the following: (Note: two_fields is FALSE)

/usr/local/bin/data/ens
cube.case
0

And you would invoke checker as:

> checker -p cube.play

Another example playfile with swapbytes 0, two enabled toggles, three pulldowns with the value 0 chosen and a single field “sample field value” could look something like the following:

/mydirectory/subdir/
myfile
0
1
1
0
0
0
sample field value

Other command line arguments are:

-server_number For checking server number routines. If you use this option, you will be prompted for the total number of servers and the current server number. These will then be used in the calls to the server number routines.

-gts # For specifying the geometry timestep to test. The default is step 0. The # is the (zero based) time step to read for geometry.

-vts # For specifying the variable timestep to test. The default is step 0. The # is the (zero based) time step to read for variables.

User Defined Writers

Users can write User Defined Writer API (UDW) to generate arbitrary data files for EnSight parts and variables. The EnSight server provides a UDW API that can be used to write out the currently selected parts in the EnSight client part list, as well as the active variables, in a user defined format. The UDW API includes methods to get, for example, node coordinates, element connectivity, ids, variable values, and time information. A UDW can call any of the methods as it wishes and create a data file(s) in any format desired. Additionally, the UDW dialog in the EnSight client has a Parameter field that provides a mechanism for passing user specified options to the UDW.

User Defined Math Functions

Users can write external variable calculator functions called User Defined Math Functions (UDMF) that can be dynamically loaded by EnSight. These functions appear in EnSight's calculator in the general function list and can be used just as any other calculator function to derive new variables.

1 User Defined Reader Version 1.0 API

This chapter will describe the EnSight User Defined Reader Version 1.0 API. Starting with EnSight Version 6.0, the 1.0 API has been available. It was designed to be friendly to those producing it (especially if a global coordinate array is a hallmark of your data format).

It does, however, require more manipulation internally by EnSight, which may require more memory and processing time. Thus, you may want to also consider the 2.0 API as described in Section 2, User Defined Reader Version 2.0 API.

If you already have a working 1.0 API reader and are happy with it - there is probably no reason to modify it to the 2.0 API unless you deal with large models and the memory use and load times are a problem, or you need any of the additional capabilities that the 2.0 API provides.

If you are producing a new reader, you should consider which will work best for your needs.

Further discussion on the philosophical differences between the two API's can be found in the Overview chapter under section, User Defined Reader APIs.

If you wish to convert an existing 1.0 API reader to the 2.0 API, see Section 2.5, Converting a 1.0 API Reader to a 2.0 API READER.

The process for producing the dynamic shared library is described in the Overview chapter under section, How To Produce A User Defined Reader.

1.1 Quick Index of Library Routines

Routine Name	Brief Description
Generally Needed for UNSTRUCTURED Data	
USERD_get_element_connectivities_for_part	part's element connectivities
USERD_get_element_ids_for_part	part's element ids
USERD_get_global_coords	global node coordinates
USERD_get_global_node_ids	global node ids
USERD_get_number_of_global_nodes	number of global nodes
USERD_get_scalar_values	global scalar variables
USERD_get_vector_values	global vector variables
Generally Needed for STRUCTURED (BLOCK) Data	
USERD_get_block_coords_by_component	block coordinates
USERD_get_block_iblanking	block iblanking values
USERD_get_block_scalar_values	block scalar variables
USERD_get_block_vector_values_by_component	block vector variables
Generally Needed for Either or Both Kinds of Data	
USERD_bkup	read/write archive routine
USERD_get_changing_geometry_status	changing geometry?
USERD_get_constant_value	constant variable's value
USERD_get_dataset_query_file_info	info about each model file
USERD_get_description_lines	file associated descrip lines
USERD_get_element_label_status	element labels?
USERD_get_name_of_reader	name of reader for GUI
USERD_get_node_label_status	node labels?
USERD_get_number_of_files_in_dataset	number of files in model
USERD_get_number_of_model_parts	number of model parts
USERD_get_number_of_time_steps	number of time steps

USERD_get_number_of_variables	number of variables
USERD_get_part_build_info	part type/descrip etc.
USERD_get_reader_descrip	provide GUI more description (optional)
USERD_get_solution_times	solution time values
USERD_get_variable_info	variable type/descrip etc.
USERD_get_variable_value_at_specific	node's or element's variable value over time
USERD_set_filenames	filenames entered in GUI
USERD_set_time_step	current time step
USERD_stop_part_building	cleanup routine
Optional Routines Added for Later Releases of EnSight	
USERD_get_extra_gui_defaults	default values for the extra GUI members
USERD_get_extra_gui_numbers	number of toggles, pulldowns and fields
USERD_get_num_xy_queries	number of xy queries
USERD_get_reader_release	release string of reader
USERD_get_var_extract_gui_defaults	default values for the var_extract members
USERD_get_var_extract_gui_numbers	number of toggles, pulldowns and fields
USERD_get_xy_query_data	gets xy query xy values
USERD_get_xy_query_info	gets xy query names, titles, num pairs, etc.
USERD_prefer_auto_distribute	tells whether reader will distribute for SOS
USERD_set_extra_gui_data	returns Extra GUI answers provided by user
USERD_set_filename_button_labels	sets Get File button text
USERD_set_var_extract_gui_data	returns var extract answers provided by user

1.2 Order Routines are Called

The various main operations are given basically in the order they will be performed. Within each operation, the order the routines will be called is given.

Called when library is loaded:

1. Setting name in the gui, and specifying one or two input fields - called when library is loaded.

USERD_get_name_of_reader
USERD_get_reader_descrip (optional)
USERD_prefer_auto_distribute (optional)
USERD_set_filename_button_labels (optional)
USERD_get_extra_gui_numbers (optional)
USERD_get_extra_gui_defaults (optional)
USERD_get_reader_release (optional)

Called once at initial data load when 'OK' pressed to load data:

2. Setting filenames and getting time info

USERD_set_extra_gui_data (optional)
USERD_set_filenames
USERD_get_number_of_time_steps
USERD_get_solution_times
USERD_set_time_step

3. Gathering info for part builder

USERD_set_time_step
USERD_get_changing_geometry_status
USERD_get_node_label_status
USERD_get_element_label_status
USERD_get_number_of_files_in_dataset
USERD_get_dataset_query_file_info
USERD_get_description_lines (for geometry)
USERD_get_number_of_model_parts
USERD_get_part_build_info
USERD_get_number_of_global_nodes
USERD_get_global_coords (for model extents)
USERD_get_block_coords_by_component (for model extents)

4. Gathering Variable info

USERD_get_number_of_variables
USERD_get_variable_info

5. Part building (per part created)

USERD_set_time_step
USERD_get_global_coords
USERD_get_global_node_ids
USERD_get_element_connectivities_for_part
USERD_get_element_ids_for_part
USERD_get_block_iblanking
USERD_get_block_coords_by_component
USERD_stop_part_building (only once when part builder dialog is closed)

6. Loading Variables

Calls for all constants at this point and not called again. (Nested loops, outer is over all constants, inner is over all timesteps):

constants:

- USERD_set_time_step
- USERD_get_constant_value

Called when activated or when timestep changes:

scalars:

- USERD_get_description_lines
- USERD_set_time_step
- USERD_get_scalar_values
- USERD_get_block_scalar_values

vectors:

- USERD_get_description_lines
- USERD_set_time_step
- USERD_get_vector_values
- USERD_get_block_vector_values_by_component

Called when timestep changes:

7. Changing geometry

changing coords only:

- USERD_set_time_step
- USERD_get_global_coords
- USERD_get_block_coords_by_component

changing connectivity:

- USERD_set_time_step
- USERD_get_number_of_model_parts
- USERD_get_part_build_info
- USERD_get_number_of_global_nodes
- USERD_get_global_coords
- USERD_get_global_node_ids
- USERD_get_element_connectivities_for_part
- USERD_get_element_ids_for_part
- USERD_get_block_iblanking
- USERD_get_block_coords_by_component

Called for node or element queries over time or at a specific node or element:

8. Node or Element queries over time

- USERD_get_variable_value_at_specific

1.3 Detailed Specifications

Include files:

The following header file is required in any file containing these library routines.

```
#include "global_extern.h"
```

And it references:

```
#include "global_extern_proto.h"
```

Basis of arrays:

Unless explicitly stated otherwise, all arrays are zero based - in true C fashion.

Global variables:

You will generally need to have a few global variables which are shared by the various library routines. The detailed specifications below have assumed the following are available. (Their names describe their purpose, and they will be used in helping describe the details of the routines below).

```
static int Numparts_available      = 0;
static int Num_unstructured_parts  = 0;
static int Num_structured_blocks   = 0;

/* Note: Numparts_available = Num_unstructured_parts + Num_structured_blocks */

static int Num_time_steps          = 1;
static int Num_global_nodes        = 0;
static int Num_variables           = 0;
static int Num_dataset_files       = 0;
static int Current_time_step       = 0;
```

Dummy (or stub) Routines:

Those routines marked optional, need not be included in a reader. They are truly optional. All other routines need to be included, but some can be dummy routines. As an example, if your data format does not have structured data, then all the routines dealing with structured (block) parts can be dummy routines.

The specifications for each routine in the API will now be given (routines are in alphabetical order):

USERD_bkup

```
/*-----------------------------------------------------------------------
 *
 *   Used in the archive process.  Save or restore info relating to
 *   your user defined reader.
 *
 *  (IN)  archive_file         = The archive file pointer
 *
 *  (IN)  backup_type          = Z_SAVE_ARCHIVE for saving archive
 *                               Z_REST_ARCHIVE for restoring archive
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Since EnSight's archive file is saved in binary form, it is
 *    suggested that you also do any writing to it or reading from it
 *    in binary.
 *
 *  * You should archive any variables, that will be needed for
 *    future operations, that will not be read or computed again
 *    before they will be needed.  These are typically global
 *    variables.
 *
 *  * Make sure that the number of bytes that you write on a save and
 *    the number of bytes that you read on a restore are identical!!
 *
 *  * And one last reminder.  If any of the variables you save are
 *    allocated arrays, you must do the allocations before restoring
 *    into them.
 *
 *  ================================
 *  * SPECIAL NOTE FOR WINDOWS ONLY:
 *    Because our current implementation under windows needs to open and close files
 *    from within the reader .dll, a special structure (named USERD_globals) needs to
 *    be defined in the global space of your reader. This structure needs to be defined
 *    like:              ------ -----
 *
 *    #ifdef WIN32                          (which includes 32 bit and 64 bit windows)
 *    W32EXPORT struct _USERD_globals {
 *      char arch_filename[256];
 *      unsigned long arch_fileptr;
 *    } USERD_globals;
 *    #endif
 *
 *    This structure will be bound when the reader .dll is loaded and will be used to
 *    store the archive file name and the current offset therein.
 *    Again for windows only, you need to ignore the archive_file pointer in the
 *    argument list and instead open and close the arch_filename file as well as keep
 *    the arch_fileptr offset current in this routine.
 *
 *    So first define the USERD_globals structure at the beginning of your reader.
 *
 *    Then, when an archive is saved, the following needs to be done in this routine:
 *      1. open USERD_globals.arch_filename for appending         (within #ifdef WIN32)
 *      2. do your writes
 *      3. close the file                                         (within #ifdef WIN32)
 *
 *    When an archive is restored, do the following in this routine:
 *      1. open USERD_globals.arch_filename for reading,
 *            and fseek to USERD_globals.arch_fileptr offset      (within #ifdef WIN32)
```

```
 *      2. do your reads
 *      3. save the new USERD_globals.arch_fileptr offset (using ftell),
 *            and close the file                                  (within #ifdef WIN32)
 *
 *    Here is some pseudo code to illustrate:
 *    ---------------------------------------
 *    switch(baskup_type) {
 *    case Z_SAVE_ARCHIVE:
 *
 *  #ifdef WIN32
 *      archive_file = fopen(USERD_globals.arch_filename,"ab");
 *  #endif
 *
 *       .
 *       .
 *       .
 *  #ifdef WIN32
 *      fclose(archive_file)
 *  #endif
 *
 *      break;
 *
 *    case Z_REST_ARCHIVE:
 *
 *  #ifdef WIN32
 *      archive_file = fopen(USERD_globals.arch_filename,"rb");
 *      fseek(archive_file, USERD_globals.arch_fileptr, SEEK_SET);
 *  #endif
 *
 *       .
 *       .
 *       .
 *
 *  #ifdef WIN32
 *      USERD_globals.arch_fileptr = ftell(archive_file);
 *      fclose(archive_file)
 *  #endif
 *
 *      break;
 *   }
 *
 *  And finally be aware of a current limitation of the
 *  Windows implementation of this routine:
 *  ---------------------------------------
 *  Because the structure uses a long for the file offset, the archive restore
 *  will not work when the offset to the information written in this routine
 *  is greater than 2 Gb, on 32 bit windows. On 64 bit windows there is no such
 *  limitation because the long is 64 bits.
 *-----------------------------------------------------------------------*/
int
USERD_bkup(FILE *archive_file,
           int backup_type)
```

USERD_get_block_coords_by_component

```
/*----------------------------------------------------------------------
 *
 *   Get the coordinates of a given block, component at a time
 *
 *  (IN)  block_number            = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that is
 *                                   loaded in USERD_get_part_build_info)
 *
 *  (IN)  which_component         = Z_COMPX if x component wanted
 *                                = Z_COMPY if y component wanted
 *                                = Z_COMPZ if z component wanted
 *
 *  (OUT) coord_array             = 1D array containing x,y, or z
 *                                   coordinate component of each node
 *
 *                                  (Array will have been allocated
 *                                   i*j*k for the block long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_structured_blocks is > 0
 *--------------------------------------------------------------------*/
int
USERD_get_block_coords_by_component(int block_number,
                                    int which_component,
                                    float *coord_array)
```

USERD_get_block_iblanking

```
/*----------------------------------------------------------------------
 *
 *   Get the iblanking value at each node of a block - If Z_IBLANKED
 *
 *  (IN)  block_number            = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                  It is NOT the part_id label that is
 *                                  loaded in USERD_get_part_build_info)
 *
 *  (OUT) iblank_array            = 1D array containing iblank value
 *                                  for each node.
 *
 *                                  (Array will have been allocated
 *                                  i*j*k for the block long)
 *
 *          possible values are:   Z_EXT     = exterior (outside)
 *                                 Z_INT     = interior (inside)
 *                                 Z_BND     = boundary
 *                                 Z_INTBND  = internal boundary
 *                                 Z_SYM     = symmetry plane
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_structured_blocks is > 0  and you have
 *    some iblanked blocks
 *----------------------------------------------------------------------*/
int
USERD_get_block_iblanking(int block_number,
                          int *iblank_array)
```

USERD_get_block_scalar_values

```
/*----------------------------------------------------------------------
 *
 *   if Z_PER_NODE:
 *     Get the values at each node of a block, for a given scalar
 *     variable.
 *
 *   or if Z_PER_ELEM:
 *     Get the values at each element of a block, for a given scalar
 *     variable.
 *
 *  (IN)  block_number              Since EnSight Version 7.4:
 *                                  -------------------------
 *                                = The block number
 *
 *                                 (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                  It is NOT the part_id label that is
 *                                  loaded in USERD_get_part_build_info)
 *
 *                                 Prior to EnSight 7.4:
 *                                 ---------------------
 *                               =  The block id label
 *
 *                                  It is the part_id label that was
 *                                  loaded in USERD_get_part_build_info
 *
 *                                  It is NOT the 1-based index of the
 *                                  part table that is used in geometry routines.
 *
 *  (IN)  which_scalar            = The variable "number" to get (1 ... Num_variables)
 *  (OUT) scalar_array            = 1D array containing scalar values
 *                                  for each node or element.
 *
 *                                  Array will have been allocated:
 *
 *                                  if Z_PER_NODE:
 *                                     i*j*k for the block long
 *
 *                                  if Z_PER_ELEM:
 *                                     (i-1)*(i-1)*(k-1) for the block long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_structured_blocks is > 0,
 *    Num_variables is > 0, and there are some scalar type variables
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *--------------------------------------------------------------------*/
int
USERD_get_block_scalar_values(int block_number,
                              int which_scalar,
                              float *scalar_array)
```

USERD_get_block_vector_values_by_component

```
/*----------------------------------------------------------------------
 *  if Z_PER_NODE:
 *    Get the values at each node of a block, for a given vector
 *    variable, one component at a time.
 *
 *  or if Z_PER_ELEM:
 *    Get the values at each element of a block, for a given vector
 *    variable, one component at a time.
 *
 *  (IN)  block_number              Since EnSight Version 7.4:
 *                                  --------------------------
 *                                = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that is
 *                                   loaded in USERD_get_part_build_info)
 *
 *                                  Prior to EnSight 7.4:
 *                                  ---------------------
 *                                =  The block id label
 *
 *                                   It is the part_id label that was
 *                                   loaded in USERD_get_part_build_info
 *
 *                                   It is NOT the 1-based index of the
 *                                   part table that is used in geometry routines.
 *
 *  (IN)  which_vector            = The variable "number" to get (1 ... Num_variables)
 *
 *  (IN)  which_component         = Z_COMPX if x component wanted
 *                                = Z_COMPY if y component wanted
 *                                = Z_COMPZ if z component wanted
 *
 *  (OUT) vector_array            = 1D array containing vector
 *                                  component value for each node or element.
 *
 *                                  Array will have been allocated:
 *
 *                                  if Z_PER_NODE:
 *                                     i*j*k for the block long
 *
 *                                  if Z_PER_ELEM:
 *                                     (i-1)*(i-1)*(k-1) for the block long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_structured_blocks is > 0,
 *    Num_variables is > 0, and there are some vector type variables
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *
 *----------------------------------------------------------------------*/
```

```
int
USERD_get_block_vector_values_by_component(int block_number,
                                           int which_vector,
                                           int which_component,
                                           float *vector_array)
```

USERD_get_changing_geometry_status

```
/*----------------------------------------------------------------------
 *
 *   Gets the changing geometry status
 *
 *  returns:  Z_STATIC        if geometry does not change
 *            Z_CHANGE_COORDS if changing coordinates only
 *            Z_CHANGE_CONN   if changing connectivity
 *
 *  Notes:
 *  * EnSight does not support changing number of parts, nor changing
 *    changing number of variables.  But the coords and/or the
 *    connectivity of the parts can change.
 *--------------------------------------------------------------------*/
int
USERD_get_changing_geometry_status( void )
```

USERD_get_constant_value

```
/*----------------------------------------------------------------------
 *
 *   Get the value of a constant at a time step
 *
 *  (IN)  which_var            = Which variable (this is the same
 *                                               implied variable
 *                                               number used in other
 *                                               functions.)
 *                                               (1 ... Num_variables)
 *
 *  returns: value of the requested constant variable
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *---------------------------------------------------------------------*/
float
USERD_get_constant_value(int which_var)
```

USERD_get_dataset_query_file_info

```
/*----------------------------------------------------------------------
 *
 *   Get the information about files in the dataset.  Used for the
 *   dataset query option.
 *
 *  (OUT) qfiles   = Structure containing information about each file
 *                   of the dataset. The Z_QFILES structure is defined
 *                   in the global_extern.h file
 *
 *                   (The structure will have been allocated
 *                    Num_dataset_files long, with 10 description
 *                    lines per file).
 *
 *      qfiles[].name        = The name of the file
 *                             (Z_MAXFILENP is the dimensioned length
 *                              of the name)
 *
 *      qfiles[].sizeb       = The number of bytes in the file
 *                             (Typically obtained with a call to the
 *                              "stat" system routine)
 *
 *      qfiles[].timemod     = The time the file was last modified
 *                             (Z_MAXTIMLEN is the dimesioned length
 *                              of this string)
 *                             (Typically obtained with a call to the
 *                              "stat" system routine)
 *
 *      qfiles[].num_d_lines = The number of description lines you
 *                              are providing from the file. Max = 10
 *
 *      qfiles[].f_desc[]    = The description line(s) per file,
 *                              qfiles[].num_d_lines of them
 *                              (Z_MAXFILENP is the allocated length of
 *                               each line)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * If Num_dataset_files is 0, this routine will not be called.
 *----------------------------------------------------------------------*/
int
USERD_get_dataset_query_file_info(Z_QFILES *qfiles)
```

USERD_get_description_lines

```
/*----------------------------------------------------------------------
 *
 *   Get two description lines associated with geometry per time step,
 *   or one description line associated with a variable per time step.
 *
 *  (IN)  which_type           = Z_GEOM for geometry
 *                             = Z_VARI for variable
 *
 *  (IN)  which_var            = If it is a variable, which one.
 *                                  (1 ... Num_variables)
 *                               Ignored if geometry type.
 *
 *  (OUT) line1                = The 1st geometry description line,
 *                               or the variable description line.
 *
 *  (OUT) line2                = The 2nd geometry description line
 *                               Not used if variable type.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * These are the lines EnSight can echo to the screen in
 *    annotation mode.
 *----------------------------------------------------------------------*/
int
USERD_get_description_lines(int which_type,
                            int which_var,
                            char line1[Z_BUFL],
                            char line2[Z_BUFL])
```

USERD_get_element_connectivities_for_part

```
/*----------------------------------------------------------------------
 *   Gets the connectivities for the elements of a part
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that
 *                                   is loaded in USERD_get_part_build_info)
 *
 *  (OUT) conn_array              = 3D array containing connectivity
 *                                   of each element of each type.
 *
 *                                  (Array will have been allocated
 *                                   Z_MAXTYPE by num_of_elements of
 *                                   each type by connectivity length
 *                                   of each type)
 *
 *                        ex) If num_of_elements[Z_TRI03] = 25
 *                               num_of_elements[Z_QUA04] = 100
 *                               num_of_elements[Z_HEX08] = 30
 *                            as obtained in:
 *                             USERD_get_part_build_info
 *
 *                            Then the allocated dimensions available
 *                            for this routine will be:
 *                               conn_array[Z_TRI03][25][3]
 *                               conn_array[Z_QUA04][100][4]
 *                               conn_array[Z_HEX08][30][8]
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_unstructured_parts is > 0
 *
 * The coord_array loaded in USERD_get_global_coords is zero-based,
 *   but within EnSight it will become a one-based array.
 *   Thus, coord_array[0] will be accessed by node 1 from the conn_array,
 *         coord_array[1] will be accessed by node 2 from the conn_array, etc.
 *      ex) Given a model of two triangles, you should load coord_array in
 *          USERD_get_global_coords as follows:
 *
 *                               node  coordinates
 *                               ----  -----------
 *          4 --------- 3         1    coord_array[0].xyz[0] = 0.0
 *           |\        |              coord_array[0].xyz[1] = 0.0
 *           | \  T2   |              coord_array[0].xyz[2] = 0.0
 *           |  \      |
 *           |   \     |         2    coord_array[1].xyz[0] = 1.0
 *           |    \    |              coord_array[1].xyz[1] = 0.0
 *           |     \   |              coord_array[1].xyz[2] = 0.0
 *           |      \  |
 *           |  T1   \ |         3    coord_array[2].xyz[0] = 1.0
 *           |        \|              coord_array[2].xyz[1] = 1.6
 *          1 --------- 2             coord_array[2].xyz[2] = 0.0
```

```
 *
 *                              4    coord_array[3].xyz[0] = 0.0
 *                                   coord_array[3].xyz[1] = 1.6
 *                                   coord_array[3].xyz[2] = 0.0
 *
 *
 *        And conn_array here as follows:
 *
 *        Triangle  Connectivity
 *        --------  ------------
 *           T1     conn_array[Z_TRI03][0][0] = 1
 *                  conn_array[Z_TRI03][0][1] = 2
 *                  conn_array[Z_TRI03][0][2] = 4
 *
 *           T2     conn_array[Z_TRI03][1][0] = 2
 *                  conn_array[Z_TRI03][1][1] = 3
 *                  conn_array[Z_TRI03][1][2] = 4
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_element_connectivities_for_part(int part_number,
                                          int **conn_array[Z_MAXTYPE])
```

USERD_get_element_ids_for_part

```
/*----------------------------------------------------------------------
 *
 *   Gets the ids for the elements of a part
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                  It is NOT the part_id label that
 *                                  is loaded in USERD_get_part_build_info)
 *
 *  (OUT) elemid_array            = 2D array containing id of each
 *                                  element of each type.
 *
 *                                  (Array will have been allocated
 *                                  Z_MAXTYPE by num_of_elements of
 *                                  each type)
 *
 *                       ex) If num_of_elements[Z_TRI03] = 25
 *                              num_of_elements[Z_QUA04] = 100
 *                              num_of_elements[Z_HEX08] = 30
 *                           as obtained in:
 *                            USERD_get_part_build_info
 *
 *                           Then the allocated dimensions available
 *                           for this routine will be:
 *                              elemid_array[Z_TRI03][25]
 *                              elemid_array[Z_QUA04][100]
 *                              elemid_array[Z_HEX08][30]
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_unstructured_parts is > 0  and element
 *    label status is TRUE
 *--------------------------------------------------------------------*/
int
USERD_get_element_ids_for_part(int part_number,
                               int *elemid_array[Z_MAXTYPE])
```

USERD_get_element_label_status

```
/*-----------------------------------------------------------------------
 *
 *  Answers the question as to whether element labels will be provided.
 *
 *  returns:  TRUE        if element labels are available
 *            FALSE       if no element labels
 *
 *  Notes:
 *  * These are needed in order to do any element querying, or
 *    element labeling on-screen.
 *
 *      For unstructured parts, you can read them from your file if
 *      available, or can assign them, etc. They need to be unique
 *      per part, and are often unique per model.
 *
 *        USERD_get_element_ids_for_part is used to obtain the ids,
 *        on a part by part basis, if TRUE status is returned here.
 *
 *      For structured parts, EnSight will assign ids if you return a
 *        status of TRUE here.  You cannot assign them youself!!
 *-----------------------------------------------------------------------*/
int
USERD_get_element_label_status( void )
```

USERD_get_extra_gui_defaults

```
/*----------------------------------------------------------------------------
 *                                                                  <optional>
 *----------------------------------------------------------------------------
 *
 *  This routine defines the Titles, status, List choices, strings, etc that
 *  are fed up to the GUI.
 *
 *   (OUT) toggle_Title              = title for each toggle
 *                                     array dimension is
 *                                [num_toggles] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) toggle_default_status     = Setting for each toggle (TRUE or FALSE)
 *                                     array dimension is [num_toggles] long
 *
 *   (OUT) pulldown_Title            = title for each pulldown
 *                                     array dimension is
 *                                [num_pulldowns] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) pulldown_number_in_list   = number of items in each pulldown
 *                                     array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_default_selection = item selection for each pulldown
 *                                     array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_item_strings     = pulldown item strings
 *                                     array is [num_pulldowns] by
 *                                            [Z_MAX_NUM_GUI_PULL_ITEMS] by
 *                                            [Z_LEN_GUI_PULL_STR] long
 *
 *   (OUT) field_Title               = title for each field
 *                                     array dimension is
 *                                 [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) field_user_string         = content of the field
 *                                     array dimension is
 *                                 [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The library is loaded, this routine is called,
 *    then the library is unloaded.
 *
 *  * Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 * ------------------------------------------------ */
int USERD_get_extra_gui_defaults(char **toggle_Title,
                                 int *toggle_default_status,
                                 char **pulldown_Title,
                                 int *pulldown_number_in_list,
                                 int *pulldown_default_selection,
                                 char ***pulldown_item_strings,
                                 char **field_Title,
                                 char **field_user_string)
```

USERD_get_extra_gui_numbers

```
/*----------------------------------------------------------------------
 *                                                              <optional>
 * ---------------------------------------------------------------------
 *
 *   The Enhanced GUI routines are added to allow the user to customize a
 *   portion of the Data Reader dialog to pass in options to their user
 *   defined reader.
 *
 *   This routine defines the numbers of toggles, pulldowns & fields
 *
 *   (OUT) num_Toggles    = number of toggles that will be provided
 *
 *   (OUT) num_pulldowns  = number of pulldowns that will be provided
 *
 *   (OUT) num_fields     = number of fields that will be provided
 *
 *   Notes:
 *   * There are three routines that work together:
 *         USERD_get_extra_gui_numbers
 *         USERD_get_extra_gui_defaults (this one)
 *         USERD_set_extra_gui_data
 *
 *     The existence of these routine indicates that
 *     you wish to add customize entries to the
 *     Data Reader dialog.
 *
 *     If you don't want the extra GUI features,
 *     simply delete these routines, or change their
 *     names to something such as
 *     USERD_DISABLED_get_extra_gui_defaults
 *
 *     The presence of these routines
 *     will ensure that EnSight will call them and
 *     use their data to modify the Data Reader dialog
 *     with some or all of the following:
 *     toggles, pulldown menu and fields.
 *
 *     The user can then interact with the enhanced
 *     GUI and then send their choices to
 *     USERD_set_extra_gui_data
 *
 *     Therefore if USERD_get_extra_gui_numbers
 *     exists then the other two must exist.
 *
 *     If none exist, then the GUI will be unchanged.
 *
 *     Toggle data will return an integer
 *                               TRUE if checked
 *                               FALSE if unchecked
 *
 *     Pulldown menu will return an integer representing
 *                               the menu item selected
 *
 *     Field will return a string Z_LEN_GUI_FIELD_STR long.
 *
 *     If all the enhanced GUI features are enabled it
 *     might look something like this
 *
 *
 *
```

```
 *     ====================================================
 *
 *        [ ] Title 1
 *        [X] Title 3
 *        [X] Title 2
 *        [X] Title 4
 *
 *        Pulldown Menu ->
 *            Menu Choice 1
 *            Menu Choice 2
 *            Menu Choice 3
 *
 *        Data Field Title 1 _________________________________
 *
 *        Data Field Title 2 _________________________________
 *
 *     ====================================================
 *
 * * The following are defined in the global_extern.h
 *          Z_MAX_NUM_GUI_PULL_ITEMS max num GUI pulldowns
 *          Z_LEN_GUI_PULL_STR  max length of GUI pulldown string
 *          Z_LEN_GUI_FIELD_STR  max length of field string
 *          Z_LEN_GUI_TITLE_STR   max length of title string
 *
 * * The library is loaded, this routine is called,
 *   then the library is unloaded.
 *
 * * Do not define globals in this routine as when the library is unloaded,
 *   you'll lose them.
 *-------------------------------------------------------------------------*/
void USERD_get_extra_gui_numbers(int *num_Toggles,
                                 int *num_pulldowns,
                                 int *num_fields)
```

USERD_get_global_coords

```
/*-------------------------------------------------------------------------
 *   Get the global coordinates
 *
 *  (OUT) coord_array             = 1D array of CRD structures,
 *                                   which contains x,y,z coordinates
 *                                   of each node.
 *
 *                                  (Array will have been allocated
 *                                   Num_global_nodes long)
 *
 * For reference, this structure (which is in global_extern) is:
 *
 *         typedef struct {
 *            float xyz[3];
 *         }CRD;
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_unstructured_parts is > 0
 *
 * The coord_array is zero-based, but within EnSight it will become
 *    a one-based array.
 *    Thus, coord_array[0] will be accessed by node 1 from the conn_array,
 *          coord_array[1] will be accessed by node 2 from the conn_array, etc.
 *
 *      ex) Given a model of two triangles, you should load coord_array as
 *          follows:
 *
 *                              node  coordinates
 *                              ----  -----------
 *          4 --------- 3        1    coord_array[0].xyz[0] = 0.0
 *           |\        |             coord_array[0].xyz[1] = 0.0
 *           | \  T2   |             coord_array[0].xyz[2] = 0.0
 *           |  \      |
 *           |   \     |        2    coord_array[1].xyz[0] = 1.0
 *           |    \    |             coord_array[1].xyz[1] = 0.0
 *           |     \   |             coord_array[1].xyz[2] = 0.0
 *           |      \  |
 *           |  T1   \ |        3    coord_array[2].xyz[0] = 1.0
 *           |        \|             coord_array[2].xyz[1] = 1.6
 *          1 --------- 2            coord_array[2].xyz[2] = 0.0
 *
 *                              4    coord_array[3].xyz[0] = 0.0
 *                                   coord_array[3].xyz[1] = 1.6
 *                                   coord_array[3].xyz[2] = 0.0
 *
 *
 *        And conn_array in USERD_get_element_connectivities_for_part
 *        as follows:
 *
 *        Triangle  Connectivity
 *        --------  ------------
 *           T1     conn_array[Z_TRI03][0][0] = 1
 *                  conn_array[Z_TRI03][0][1] = 2
 *                  conn_array[Z_TRI03][0][2] = 4
```

```
 *
 *            T2     conn_array[Z_TRI03][1][0] = 2
 *                   conn_array[Z_TRI03][1][1] = 3
 *                   conn_array[Z_TRI03][1][2] = 4
 *
 *-------------------------------------------------------------------------*/
int
USERD_get_global_coords(CRD *coord_array)
```

USERD_get_global_node_ids

```
/*----------------------------------------------------------------------
 *
 *   Get the global nodeids
 *
 *  (OUT) nodeid_array            = 1D array containing node ids of
 *                                   each node. The ids must be > 0
 *
 *                                 (Array will have been allocated
 *                                  Num_global_nodes long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_unstructured_parts is > 0  and
 *    node label status is TRUE
 *--------------------------------------------------------------------*/
int
USERD_get_global_node_ids(int *nodeid_array)
```

USERD_get_name_of_reader

```
/*----------------------------------------------------------------------
 *
 *   Gets the name of your user defined reader.  The user interface will
 *   ask for this and include it in the available reader list.
 *
 *  (OUT) reader_name          = the name of the reader (data format)
 *                             (max length is Z_MAX_USERD_NAME, which
 *                              is 20)
 *
 *  (OUT) *two_fields          = FALSE if only one data field is
 *                                      required.
 *                               TRUE if two data fields required
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Always called.  Provide a name for your custom reader format
 *
 *  * If you don't want a custom reader to show up in the data dialog
 *    choices, return a name of "No_Custom"
 *----------------------------------------------------------------------*/
int
USERD_get_name_of_reader(char reader_name[Z_MAX_USERD_NAME],
                         int *two_fields)
```

USERD_get_node_label_status

```
/*----------------------------------------------------------------------
 *
 *   Answers the question as to whether node labels will be provided
 *
 *  returns:  TRUE        if node labels are available
 *            FALSE       if no node labels
 *
 *  Notes:
 *  * These are needed in order to do any node querying, or node
 *    labeling on-screen                                    .
 *
 *      For unstructured parts, you can read them from your file if
 *      available, or can assign them, etc. They need to be unique
 *      per part, and are often unique per model. They must also be
 *       positive numbers greater than zero.
 *
 *        USERD_get_global_node_ids is used to obtain the ids, if the
 *        status returned here is TRUE.
 *
 *        Also be aware that if you say node labels are available,
 *        the connectivity of elements must be according to these
 *        node ids.
 *
 *      For structured parts, EnSight will assign ids if you return a
 *        status of TRUE here.  You cannot assign them yourself!!
 *----------------------------------------------------------------------*/
int
USERD_get_node_label_status( void )
```

USERD_get_num_xy_queries

```
/*-----------------------------------------------------------------------
 *                                                              <optional>
 *-----------------------------------------------------------------------
 *
 *   Get the total number of xy queries in the dataset.
 *
 *  returns: the total number of xy queries in the dataset
 *
 *  Notes:
 *  * You can be as complete as you want about this.  If you don't
 *    care about xy queries, return a value of 0
 *    If you only want certain xy queries, you can just include them. But,
 *    you will need to supply the info and data USERD_get_xy_query_info
 *    and USERD_get_xy_query_data for each xy query you include here.
 *-----------------------------------------------------------------------*/
int
USERD_get_num_xy_queries( void )
```

USERD_get_number_of_files_in_dataset

```
/*----------------------------------------------------------------------
 *
 *   Get the total number of files in the dataset.  Used for the
 *   dataset query option.
 *
 *  returns: the total number of files in the dataset
 *
 *  Notes:
 *  * You can be as complete as you want about this.  If you don't
 *    care about the dataset query option, return a value of 0
 *    If you only want certain files, you can just include them. But,
 *    you will need to supply the info in USERD_get_dataset_query_file_info
 *    for each file you include here.
 *
 *  * Num_dataset_files would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_files_in_dataset( void )
```

USERD_get_number_of_global_nodes

```
/*----------------------------------------------------------------------
 *
 *   Gets the number of global nodes, used for unstructured parts
 *
 *  returns: number of global nodes (>=0 if okay, <0 if problems)
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * For unstructured data:
 *         EnSight wants  1. A global array of nodes
 *                        2. Element connectivities by part, which
 *                           reference the node numbers of the global
 *                           node array.
 *              IMPORTANT:
 *              ---------
 *              If you provide node ids, then element connectivities
 *              must be in terms of the node ids.  If you do not
 *              provide node ids, then element connectivities must be
 *              in terms of the index into the node array, but shifted
 *              to start at 1
 *
 *  * Not called unless Num_unstructured_parts is > 0
 *
 *  * Num_global_nodes would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_global_nodes( void )
```

USERD_get_number_of_model_parts

```
/*----------------------------------------------------------------------
 *
 *   Gets the total number of unstructured and structured parts
 *   in the model, for which you can supply information.
 *
 *  returns:  num_parts  (>0 if okay, <=0 if probs)
 *
 *  Notes:
 *  * If going to have to read down through the parts in order to
 *    know how many, you may want to build a table of pointers to
 *    the various parts, so can easily get to particular parts in
 *    later processes.  If you can simply read the number of parts
 *    at the head of the file, then you would probably not build the
 *    table at this time.
 *
 *  * This routine would set Numparts_available, which is equal to
 *     Num_unstructured_parts + Num_structured_blocks.
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_model_parts( void )
```

USERD_get_number_of_time_steps

```
/*----------------------------------------------------------------------
 *
 *   Get the number of time steps of data available.
 *
 *  returns: number of time steps (>0 if okay, <=0 if problems).
 *
 *  Notes:
 *  * This should be >= 1       1 indicates a static problem
 *                             >1 indicates a transient problem
 *
 *  * Num_time_steps would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_time_steps( void )
```

USERD_get_number_of_variables

```
/*----------------------------------------------------------------------
 *
 *   Get the number of variables for which you will be providing info.
 *
 *  returns: number of variables (includes constant, scalar, vector,
 *            and tensor types)
 *            >=0 if okay
 *            <0 if problem
 *
 *  Notes:
 *  * Variable numbers, by which references will be made, are implied
 *    here. If you say there are 3 variables, the variable numbers
 *    will be 1, 2, and 3.
 *
 *  * Num_variables would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_variables( void )
```

USERD_get_part_build_info

```
/*----------------------------------------------------------------------
 *
 *   Gets the info needed for part building process
 *
 *  (OUT) part_id                 = Array containing part ids for
 *                                  each of the model parts.
 *
 *                                  IMPORTANT:
 *                                   Parts ids must be >= 1, because
 *                                   of the way the GUI uses them
 *
 *              example:  If Numparts_available = 3   (num_parts in the
 *                                                     USERD_get_number_of_model_parts
 *                                                     routine)
 *
 *                        table index        part_id
 *                        -----------        -------
 *                         1                  13
 *                         2                  57
 *                         3                  125
 *
 *             ****************************************
 *              Previous to version 7.4 of EnSight, there is
 *              an inconsistency in the way that parts are
 *              referenced in the arguments to various routines
 *              in this API.  This inconsistency doesn't matter
 *              whenever your parts are 1,2,3,...  And thus
 *              most of you have never noticed the problem.
 *
 *              The ids provided here are the numbers by
 *              which the parts will be referred to in the
 *              GUI (if possible). Starting with EnSight
 *              version 7.4, they are treated only as labels
 *              in the GUI.
 *
 *               Starting with EnSight 7.4, all routines which have "part_number",
 *               "block_number", or "which_part" as arguments - are expecting the
 *               table index (1,2,3).
 *
 *               Prior to EnSight 7.4, the arguments "part_number", "block_number",
 *               or "which_part" refer to:
 *                 the table index (1,2,3) for the following routines:
 *                       USERD_get_element_connectivities_for_part
 *                       USERD_get_element_ids_for_part
 *                       USERD_get_block_coords_by_component
 *                       USERD_get_block_iblanking
 *
 *                 but, to the part_id labels (12,57,125) for the following routines:
 *                       USERD_get_scalar_values
 *                       USERD_get_vector_values
 *                       USERD_get_block_scalar_values
 *                       USERD_get_block_vector_values_by_component
 *                       USERD_get_variable_value_at_specific
 *             ****************************************
 *
 *                                (Array will have been allocated
 *                                 Numparts_available long)
 *
 *
 *
```

```
*  (OUT) part_types             = Array containing one of the
*                                 following for each model part:
*
*                                      Z_UNSTRUCTURED or
*                                      Z_STRUCTURED  or
*                                      Z_IBLANKED
*
*                                 (Array will have been allocated
*                                  Numparts_available long)
*
*  (OUT) part_description       = Array containing a description
*                                 for each of the model parts
*
*                                 (Array will have been allocated
*                                  Numparts_available by Z_BUFL long)
*
*  (OUT) number_of_elements     = 2D array containing number of
*                                 each type of element for each
*                                 unstructured model part.
*                                 ------------
*                                 Possible types are:
*
*                                  Z_POINT   =  point
*                                  Z_BAR02   =  2-noded bar
*                                  Z_BAR03   =  3-noded bar
*                                  Z_TRI03   =  3-noded triangle
*                                  Z_TRI06   =  6-noded triangle
*                                  Z_QUA04   =  4-noded quadrilateral
*                                  Z_QUA08   =  8-noded quadrilateral
*                                  Z_TET04   =  4-noded tetrahedron
*                                  Z_TET10   = 10-noded tetrahedron
*                                  Z_PYR05   =  5-noded pyramid
*                                  Z_PYR13   = 13-noded pyramid
*                                  Z_PEN06   =  6-noded pentahedron
*                                  Z_PEN15   = 15-noded pentahedron
*                                  Z_HEX08   =  8-noded hexahedron
*                                  Z_HEX20   = 20-noded hexahedron
*
*                                (Ignored unless Z_UNSTRUCTURED type)
*
*                                 (Array will have been allocated
*                                  Numparts_available by
*                                  Z_MAXTYPE long)
*
*  (OUT) ijk_dimensions         = 2D array containing ijk dimensions
*                                 for each structured model part.
*                                           ----------
*                                 (Ignored if Z_UNSTRUCTURED type)
*
*                                 (Array will have been allocated
*                                  Numparts_available by 3 long)
*
*                               ijk_dimensions[][0] = I dimension
*                               ijk_dimensions[][1] = J dimension
*                               ijk_dimensions[][2] = K dimension
*
*  (OUT) iblanking_options      = 2D array containing iblanking
*                                 options possible for each
*                                 structured model part.
*                                 ----------
*                                 (Ignored unless Z_IBLANKED type)
```

```
 *
 *                                  (Array will have been allocated
 *                                   Numparts_available by 6 long)
 *
 *      iblanking_options[][Z_EXT]     = TRUE if external (outside)
 *                       [][Z_INT]     = TRUE if internal (inside)
 *                       [][Z_BND]     = TRUE if boundary
 *                       [][Z_INTBND]  = TRUE if internal boundary
 *                       [][Z_SYM]     = TRUE if symmetry surface
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * If you haven't built a table of pointers to the different parts,
 *    you might want to do so here as you gather the needed info.
 *
 *  * This will be based on Current_time_step
 *-------------------------------------------------------------------------*/
int
USERD_get_part_build_info(int *part_id,
                          int *part_types,
                          char *part_description[Z_BUFL],
                          int *number_of_elements[Z_MAXTYPE],
                          int *ijk_dimensions[3],
                          int *iblanking_options[6])
```

USERD_get_reader_descrip

```
/*----------------------------------------------------------------------
 *                                                              <optional>
 *
 *  Gets the description of the reader, so gui can give more info
 *
 *  (OUT) reader_descrip       = the description of the reader
 *                              (max length is MAXFILENP, which
 *                               is 255)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * OPTIONAL ROUTINE!
 *--------------------------------------------------------------------*/
int
USERD_get_reader_descrip(char descrip[Z_MAXFILENP])
```

USERD_get_reader_release

```
/*----------------------------------------------------------------------
 *                                                              <optional>
  *
 *   Gets the release string for the reader.
 *
 *   This release string is a free-format string which is for
 *   informational purposes only.  It is often useful to increment
 *   the release number/letter to indicate a change in the reader.
 *   The given string will simply be output by the EnSight server
 *   when the reader is selected.
 *
 *  (OUT) release_number       = the release number of the reader
 *                             (max length is Z_MAX_USERD_NAME, which
 *                              is 20)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Called when the reader is selected for use.
 *----------------------------------------------------------------------*/
int
USERD_get_reader_release(char version_number[Z_MAX_USERD_NAME])
```

USERD_get_scalar_values

```
/*----------------------------------------------------------------------
 *
 * if Z_PER_NODE:
 *   Get the values at each global node for a given scalar variable.
 *
 * or if Z_PER_ELEM:
 *   Get the values at each element of a specific part and type for a
 *   given scalar variable.
 *
 *  (IN)  which_scalar            = The variable "number" to get
 *                                      (1 ... Num_variables)
 *
 *  (IN)  which_part
 *
 *           if Z_PER_NODE:         Not used
 *
 *           if Z_PER_ELEM:         Since EnSight Version 7.4:
 *                                  -------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that
 *                                   is loaded in USERD_get_part_build_info)
 *
 *
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label loaded
 *                                                in USERD_get_part_build_info. It is
 *                                                NOT the part table index.
 *
 * (IN)  which_type
 *
 *           if Z_PER_NODE:         Not used
 *
 *           if Z_PER_ELEM:       = The element type
 *
 *  (OUT) scalar_array
 *
 *            if Z_PER_NODE:       = 1D array containing scalar values
 *                                   for each node.
 *
 *                                   (Array will have been allocated
 *                                    Num_global_nodes long)
 *
 *            if Z_PER_ELEM:       = 1d array containing scalar values for
 *                                   each element of a particular part & type.
 *
 *                                  (Array will have been allocated
 *                                   number_of_elements[which_part][which_type]
 *                                    long.  See USERD_get_part_build_info)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
```

```
 *
 *  * Not called unless Num_unstructured_parts is > 0,
 *    Num_variables is > 0, and you have some scalar type variables
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_scalar_values(int which_scalar,
                        int which_part,
                        int which_type,
                        float *scalar_array)
```

USERD_get_solution_times

```
/*----------------------------------------------------------------------
 *
 *   Get the solution times associated with each time step.
 *
 *  (OUT) solution_times       = 1D array of solution times/time step
 *
 *                                  (Array will have been allocated
 *                                   Num_time_steps long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * These must be non-negative and increasing.
 *----------------------------------------------------------------------*/
int
USERD_get_solution_times(float *solution_times)
```

USERD_get_var_extract_gui_defaults

```
/*-----------------------------------------------------------------------
 *                                                                <optional>
 *
 *  This routine defines the Titles, status, List choices, strings, etc that
 *  are fed up to the GUI for that after read situation.  (It is very similar
 *  to the USERD_get_extra_gui_defaults routine, which occurs before the read)
 *
 *   (OUT) toggle_Title              = title for each toggle
 *                                     array dimension is
 *                                [num_toggles] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) toggle_default_status     = Setting for each toggle (TRUE or FALSE)
 *                                     array dimension is [num_toggles] long
 *
 *   (OUT) pulldown_Title            = title for each pulldown
 *                                     array dimension is
 *                                [num_pulldowns] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) pulldown_number_in_list   = number of items in each pulldown
 *                                     array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_default_selection = item selection for each pulldown
 *                                     array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_item_strings     = pulldown item strings
 *                                     array is [num_pulldowns] by
 *                                            [Z_MAX_NUM_GUI_PULL_ITEMS] by
 *                                            [Z_LEN_GUI_PULL_STR] long
 *
 *   (OUT) field_Title               = title for each field
 *                                     array dimension is
 *                                [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) field_user_string         = content of the field
 *                                     array dimension is
 *                                [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The library is loaded, this routine is called,
 *    then the library is unloaded.
 *
 *  * Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 * ---------------------------------------------- */
int USERD_get_var_extract_gui_defaults(char **toggle_Title,
                                       int *toggle_default_status,
                                       char **pulldown_Title,
                                       int *pulldown_number_in_list,
                                       int *pulldown_default_selection,
                                       char ***pulldown_item_strings,
                                       char **field_Title,
                                       char **field_user_string)
```

USERD_get_var_extract_gui_numbers

```
/*----------------------------------------------------------------------
 *                                                              <optional>
 *
 *   The Var_Extract GUI routines are added to allow the user to customize a
 *   extraction parameters for variable "after" the file has been read.
 *   These things can be modified and the variables will be updated/refreshed
 *   according to the new parameters.
 *   (It is similar to the USERD_get_extra_gui_numbers routine)
 *
 *   This routine defines the numbers of toggles, pulldowns & fields
 *
 *   (OUT) num_Toggles    = number of toggles that will be provided
 *
 *   (OUT) num_pulldowns  = number of pulldowns that will be provided
 *
 *   (OUT) num_fields     = number of fields that will be provided
 *
 *   Notes:
 *   * There are three routines that work together:
 *         USERD_get_var_extract_gui_numbers
 *         USERD_get_var_extract_gui_defaults
 *         USERD_set_var_extract_gui_data
 *
 *     The existence of these routine indicates that
 *     you wish to have the Var Extract capability.
 *
 *     If you don't want the Var Extract GUI features,
 *     simply delete these routines, or change their
 *     names to something such as
 *     USERD_DISABLED_get_var_extract_gui_defaults
 *
 *     The presence of these routines
 *     will ensure that EnSight will call them and
 *     use their data to modify the extraction parameters
 *     with some or all of the following:
 *     toggles, pulldown menu and fields.
 *
 *     The user can then interact with the var extract portion of the
 *     GUI and then send their choices to
 *     USERD_set_var_extract_gui_data
 *
 *     Therefore if USERD_get_var_extract_gui_numbers
 *     exists then the other two must exist.
 *
 *     If none exist, then the GUI will be unchanged.
 *
 *     Toggle data will return an integer
 *                               TRUE if checked
 *                               FALSE if unchecked
 *
 *     Pulldown menu will return an integer representing
 *                               the menu item selected
 *
 *     Field will return a string Z_LEN_GUI_FIELD_STR long.
 *
 * * The following are defined in the global_extern.h
 *          Z_MAX_NUM_GUI_PULL_ITEMS max num GUI pulldowns
 *          Z_LEN_GUI_PULL_STR  max length of GUI pulldown string
 *          Z_LEN_GUI_FIELD_STR  max length of field string
 *          Z_LEN_GUI_TITLE_STR   max length of title string
```

```
 *
 * * The library is loaded, this routine is called,
 *   then the library is unloaded.
 *
 * *  Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 *-------------------------------------------------------------------------------*/
void USERD_get_var_extract_gui_numbers(int *num_Toggles,
                                       int *num_pulldowns,
                                       int *num_fields)
```

USERD_get_variable_info

```
/*----------------------------------------------------------------------
  *   Get the variable descriptions, types and filenames
 *
 *  (OUT) var_description      = Variable descriptions
 *
 *                               (Array will have been allocated
 *                                Num_variables by Z_BUFL long)
 *
 *          variable description restrictions:
 *          ----------------------------------
 *          1. Only first 19 characters used in EnSight.
 *          2. Leading and trailing whitespace will be removed by EnSight.
 *          3. Illegal characters will be replaced by underscores.
 *          4. Thay may not start with a numeric digit.
 *          4. No two variables may have the same description.
  *
 *  (OUT) var_filename         = Variable filenames
 *
 *                               (Array will have been allocated
 *                                Num_variables by Z_BUFL long)
 *
 *  (OUT) var_type             = Variable type
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *                              types are:  Z_CONSTANT
 *                                          Z_SCALAR
 *                                          Z_VECTOR
 *
 *  (OUT) var_classify         = Variable classification
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *                              types are:  Z_PER_NODE
 *                                          Z_PER_ELEM
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The implied variable numbers apply, but be aware that the
 *    arrays are zero based.
 *    So for variable 1, will need to provide   var_description[0]
 *                                              var_filename[0]
 *                                              var_type[0]
 *                                              var_classify[0]
 *
 *       for variable 2, will need to provide   var_description[1]
 *                                              var_filename[1]
 *                                              var_type[1]
 *                                              var_classify[1]
 *            etc.
 *--------------------------------------------------------------------*/
int
USERD_get_variable_info(char **var_description,
                        char **var_filename,
                        int *var_type,
                        int *var_classify)
```

USERD_get_variable_value_at_specific

```
/*----------------------------------------------------------------------
 *  if Z_PER_NODE:
 *    Get the value of a particular variable at a particular node in a
 *    particular part at a particular time.
 *
 *  or if Z_PER_ELEM:
 *    Get the value of a particular variable at a particular element of
 *    a particular type in a particular part at a particular time.
 *
 *  (IN)  which_var   = Which variable (1 ... Num_variables)
 *
 *  (IN)  which_node_or_elem
 *
 *                      If Z_PER_NODE:
 *                        = The node number.  This is not the id, but is
 *                                            the index of the global node
 *                                            list (1 based), or the block's
 *                                            node list (1 based).
 *
 *                            Thus,  coord_array[1]
 *                                   coord_array[2]
 *                                   coord_array[3]
 *                                        .      |
 *                                        .      |which_node_or_elem index
 *                                        .             ----
 *
 *
 *                      If Z_PER_ELEM:
 *                        = The element number. This is not the id, but is
 *                                              the element number index
 *                                              of the number_of_element array
 *                                              (see USERD_get_part_build_info),
 *                                              or the block's element list
 *                                              (1 based).
 *
 *                            Thus,  for which_part:
 *                                   conn_array[which_elem_type][0]
 *                                   conn_array[which_elem_type][1]
 *                                   conn_array[which_elem_type][2]
 *                                        .                      |
 *                                        .          (which_node_or_elem - 1) index
 *                                        .                          ----
 *
 *
 *  (IN)  which_part
 *
 *                        If Z_PER_NODE, or block part:
 *                          = Not used
 *
 *                        If Z_PER_ELEM:
 *                                  Since EnSight Version 7.4:
 *                                  --------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that
 *                                   is loaded in USERD_get_part_build_info)
```

```
 *
 *
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label loaded
 *                                                in USERD_get_part_build_info. It is
 *                                                NOT the part table index.
 *
 *  (IN)  which_elem_type
 *
 *                        If Z_PER_NODE, or block part:
 *                          = Not used
 *
 *                        If Z_PER_ELEM:
 *                          = The element type.  This is the element type index
 *                                               of the number_of_element array
 *                                               (see USERD_get_part_build_info)
 *
 *  (IN)  time_step   = Time step to use (0 to Num_time_steps-1)
 *
 *  (OUT) values      = scalar or vector component value(s)
 *                        values[0] = scalar or vector[0]
 *                        values[1] = vector[1]
 *                        values[2] = vector[2]
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *           Z_NOT_IMPLEMENTED if not implemented and want to use the slower,
 *                             complete update method within EnSight.
 *
 *  Notes:
 *  * This routine is used in node querys over time (or element querys over
 *    time for Z_PER_ELEM variables).  If these operations are not critical
 *    to you, this can be a dummy routine.
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *-------------------------------------------------------------------------*/
int
USERD_get_variable_value_at_specific(int which_var,
                                     int which_node_or_elem,
                                     int which_part,
                                     int which_elem_type,
                                     int time_step,
                                     float values[3])
```

USERD_get_vector_values

```
/*----------------------------------------------------------------------
 *
 *  if Z_PER_NODE:
 *    Get the values at each global node for a given vector variable.
 *
 *  or if Z_PER_ELEM:
 *    Get the values at each element of a specific part and type for a
 *    given vector variable.
 *
 *  (IN)  which_vector            = The variable "number" to get (1 ... Num_variables)
 *
 *  (IN)  which_part
 *
 *           if Z_PER_NODE:         Not used
 *
 *           if Z_PER_ELEM:         Since EnSight Version 7.4:
 *                                  --------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                      1 ... Numparts_available.
 *
 *                                   It is NOT the part_id label that
 *                                   is loaded in USERD_get_part_build_info)
 *
 *
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label loaded
 *                                                in USERD_get_part_build_info. It is
 *                                                NOT the part table index.
 *
 *  (IN)  which_type
 *
 *           if Z_PER_NODE:         Not used
 *
 *           if Z_PER_ELEM:       = The element type
 *
 * (OUT) vector_array
 *
 *          if Z_PER_NODE:       = 1D array containing vector values
 *                                   for each node.
 *
 *                                 (Array will have been allocated
 *                                  3 by Num_global_nodes long)
 *
 *                     Info stored in this fashion:
 *                           vector_array[0] = xcomp of node 1
 *                           vector_array[1] = ycomp of node 1
 *                           vector_array[2] = zcomp of node 1
 *
 *                           vector_array[3] = xcomp of node 2
 *                           vector_array[4] = ycomp of node 2
 *                           vector_array[5] = zcomp of node 2
 *
 *                           vector_array[6] = xcomp of node 3
 *                           vector_array[7] = ycomp of node 3
 *                           vector_array[8] = zcomp of node 3
 *                           etc.
```

```
 *
 *           if Z_PER_ELEM:      = 1d array containing vector values for
 *                                 each element of a particular part and type.
 *
 *                               (Array will have been allocated
 *                               3 by number_of_elements[which_part][which_type]
 *                               long.  See USERD_get_part_build_info)
 *
 *                     Info stored in this fashion:
 *                          vector_array[0] = xcomp of elem 1 (of part and type)
 *                          vector_array[1] = ycomp of elem 1         "
 *                          vector_array[2] = zcomp of elem 1         "
 *
 *                          vector_array[3] = xcomp of elem 2         "
 *                          vector_array[4] = ycomp of elem 2         "
 *                          vector_array[5] = zcomp of elem 2         "
 *
 *                          vector_array[6] = xcomp of elem 3         "
 *                          vector_array[7] = ycomp of elem 3         "
 *                          vector_array[8] = zcomp of elem 3         "
 *                          etc.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_unstructured_parts is > 0,
 *    Num_variables is > 0,  and you have some vector type variables
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_vector_values(int which_vector,
                        int which_part,
                        int which_type,
                        float *vector_array)
```

USERD_get_xy_query_data

```
/*----------------------------------------------------------------------------
 *                                                                  <optional>
 *
 * Gets the xy values for a particular xy_query
 *
 * (IN)  query_num            = query number (zero based)
 *                              (0 to one less than the number of querys
 *                               returned in USERD_get_num_xy_queries)
 *
 * (IN)  num_vals             = number of xy pairs in the query
 *
 * (OUT) xvals                = array of x values
 *
 * (OUT) yvals                = array of y values
 *
 * returns: Z_OK  if successful
 *          Z_ERR if a problem
 *
 * Notes:
 *----------------------------------------------------------------------------*/
int USERD_get_xy_query_data(
        int query_num,
        int num_vals,
        float *xvals,
        float *yvals)
```

USERD_get_xy_query_info

```
/*----------------------------------------------------------------------------
 *                                                                  <optional>
 *
 * Gets name, axis titles, and number of xy pairs for a particular xy_query
 *
 * (IN)  query_num            = query number (zero based)
 *                              (0 to one less than the number of querys
 *                               returned in USERD_get_num_xy_queries)
 *
 * (OUT) query_name           = Name for the xy query. (80 chars long)
 *
 * (OUT) query_xtitle         = Title for x axis       (80 chars long)
 *
 * (OUT) query_ytitle         = Title for y axis       (80 chars long)
 *
 * (OUT) query_num_pairs      = number of xy pairs
 *
 * returns: Z_OK  if successful
 *          Z_ERR if a problem
 *
 * Notes:
 *----------------------------------------------------------------------------*/
int USERD_get_xy_query_info(int query_num,
                            char *query_name,
                            char *query_xtitle,
                            char *query_ytitle,
                            int *query_num_pairs )
```

USERD_prefer_auto_distribute

```
/*----------------------------------------------------------------------
 *                                                              <optional>
 *
 * Returns whether the reader will do its own partitioning for SOS
 *
 * returns: FALSE if prefers to do its own partitioning for SOS
 *          TRUE  if EnSight will be asked to do the partitioning
 *                if an auto-distribute is specified
 *
 * Notes:
 *----------------------------------------------------------------------*/
int
USERD_prefer_auto_distribute(void) {
```

USERD_set_extra_gui_data

```
/*----------------------------------------------------------------------------
 *                                                                    <optional>
 *
 *  Receives the toggle, pulldown and field_text from enhanced GUI.
 *
 *  (IN) toggle values   TRUE = toggle checked
 *                       FALSE = toggle unchecked
 *                       Is num_Toggles long, as set in
 *                         USERD_get_extra_gui_numbers
 *
 *  (IN) pulldown value  from 0 to number of pulldown values
 *                       Is num_pulldowns long, as set in
 *                         USERD_get_extra_gui_numbers
 *
 *  (IN) field text      any text
 *                      '\0' if inactivated or nothing entered
 *                       Is num_fields by Z_LEN_GUI_FIELD_STR, as set in
 *                       USERD_get_extra_gui_numbers
 *
 *  Notes:
 *  This routine is called when the library is permanently
 *   loaded to the EnSight session, so define your globals
 *   in this and later routines.
 *
 *  It's up to you to change your reader behavior according to
 *    user entries!
 * ------------------------------------------------------------- */
void
USERD_set_extra_gui_data(int *toggle,
                         int *pulldown,
                         char **field_text)
```

USERD_set_filename_button_labels

```
/*----------------------------------------------------------------------
 *                                                              <optional>
 *
 *   Returns the labels that the EnSight GUI will place on the buttons
 *   in the Data Reader/Open dialog for Geometry and Results
 *
 *   (OUT) filename_label_1  = Label for the first button
 *                             (Z_MAX_USERD_NAME long)
 *                                (generally the geom file)
 *
 *   (OUT) filename_label_2  = Label for the second button
 *                            (Z_MAX_USERD_NAME long)
 *                                (generally the results file)
 *                             Not needed (so can be null) if two_fields
 *                             is FALSE in USERD_get_name_of_reader
 *
 *  Notes:
 *----------------------------------------------------------------------*/
void
USERD_set_filename_button_labels(char filename_label_1[Z_MAX_USERD_NAME],
                                 char filename_label_2[Z_MAX_USERD_NAME])
```

USERD_set_filenames

```
/*----------------------------------------------------------------------
 *
 *   Receives the geometry and second text field entered in the data
 *   dialog.  The user written code will have to store and use these
 *   as needed.  The user written code must manage its own files!!
 *
 *  (IN) filename_1   = the filename entered into the geometry
 *                         field of the data dialog.
 *
 *  (IN) filename_2   = The usage of this string depends on
 *                      'two_fields' in USERD_get_name_of_reader.
 *
 *                      If two_fields is FALSE then it's empty.
 *
 *                      If two_fields is TRUE, this is the
 *                      manditory results file entered
 *                      into the result field of the data dialog.
 *
 *  (IN) the_path     = the path info from the data dialog.
 *                      Note: filename_1 and filename_2 have already
 *                            had the path prepended to them.  This
 *                            is provided in case it is needed for
 *                            filenames contained in one of the files
 *
 *  (IN) swapbytes    = TRUE if should swap bytes
 *                    = FALSE normally
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Since you must manage everything from the input that is entered in
 *    these data dialog fields, this is an important routine!
 *
 *  * Since you manage these files, they can be whatever.  Perhaps
 *    you will use only one, and have references to everything else
 *    you need within it, like EnSight6 does.
 *----------------------------------------------------------------------*/
int
USERD_set_filenames(char filename_1[],
                    char param_2[],
                    char the_path[],
                    int swapbytes)
```

USERD_set_time_step

```
/*-------------------------------------------------------------------------
 *
 *   Set the current time step.  All functions that need time, and
 *   that do not explicitly pass it in, will use this time step if
 *   needed.
 *
 *  (IN) time_step - The current time step (0 to Num_time_steps-1)
 *
 * Note:
 *  * Current_time_step would be set here
 *
 *  * This routine is called from the server exit_rout with a -1
 *    argument.  This is the chance to clean up anything that
 *    should be cleaned up upon exit.  Like temporary files....
 *-------------------------------------------------------------------------*/
void
USERD_set_time_step(int time_step)
```

USERD_set_var_extract_gui_data

```
/*----------------------------------------------------------------------------
 *                                                                  <optional>
 *
 *  Receives the toggle, pulldown and field_text from var extract input.
 *
 *  (IN) toggle values   TRUE = toggle checked
 *                       FALSE = toggle unchecked
 *                       Is num_Toggles long, as set in
 *                         USERD_get_var_extract_gui_numbers
 *
 *  (IN) pulldown value  from 0 to number of pulldown values
 *                       Is num_pulldowns long, as set in
 *                         USERD_get_var_extract_gui_numbers
 *
 *  (IN) field text      any text
 *                      '\0' if inactivated or nothing entered
 *                       Is num_fields by Z_LEN_GUI_FIELD_STR, as set in
 *                       USERD_get_var_extract_gui_numbers
 *
 *  Notes:
 *  This routine is called when the library is permanently
 *   loaded to the EnSight session, so define your globals
 *   in this and later routines.
 *
 *  It's up to you to change your reader behavior according to
 *    user entries!
 * ------------------------------------------------------------- */
void
USERD_set_var_extract_gui_data(int *toggle,
                               int *pulldown,
                               char **field_text)
```

USERD_stop_part_building

```
/*----------------------------------------------------------------------
 *
 *   This routine called when the part building dialog is closed.  It is
 *   provided in case you desire to release memory, etc. that was only needed
 *   during the part building process.
 *---------------------------------------------------------------------*/
void
USERD_stop_part_building( void )
```

2 User Defined Reader Version 2.0 API

This chapter will describe the EnSight User Defined Reader Version 2.0 API. It was designed to be considerably more efficient than the 1.0 API and has a number of new features that later versions of EnSight can take advantage of.

If you are producing a new reader, or considering upgrading an existing version 1.0 API reader, please see the discussion on the philosophical differences between the two API's in the Overview chapter under section, User Defined Reader APIs.

If you wish to convert an existing 1.0 API reader to the 2.0 API, see Section 2.5, Converting a 1.0 API Reader to a 2.0 API READER.

The process for producing the dynamic shared library is described in the Overview chapter under section, How To Produce A User Defined Reader.

2.1 Quick Index of 2.0 Library Routines

Routine Name	Optional	Routine Description
Generally Needed for UNSTRUCTURED data		
USERD_get_part_coords		Part's node coordinates
USERD_get_part_coords_in_buffers	X	Part's node coordinates in buffers. (For unstructured autodistrib for SOS)
USERD_get_part_elements_by_type		Part's element connectivities
USERD_get_part_elements_by_type_in_buffers	X	Part's element connectivities in buffers. (For unstructured autodistrib for SOS)
USERD_get_periodic_ghosts_num_pairs	X	Return the number of pairs of nodes for each part periodic symmetry face
USERD_get_periodic_ghosts_pairs	X	Return the pairs for each symmetry face
Generally Needed for BLOCK data		
USERD_get_block_coords_by_component		Block node coordinates
USERD_get_block_ghost_flags		Block ghost cell flags
USERD_get_block_iblanking		Block iblanking values
USERD_set_block_range_and_stride		Sets the min, max, and stride of a block (if doing structured cinching)
USERD_get_ghosts_in_block_flag		Block ghost cell existence?
USERD_get_periodic_ghosts_structured_face_info	X	Returns face info for structured parts with periodic symmetry
Generally needed for either or both kinds of data		
USERD_bkup		Archive routine
USERD_exit_routine		Cleanup upon exit routine
USERD_get_border_availability		Part border provided?
USERD_get_border_elements_by_type		Part border conn & parent info
USERD_get_changing_geometry_status		Changing geometry?
USERD_get_constant_val		Constant variable's value
USERD_get_dataset_query_file_info		Info about each model file
USERD_get_descrip_lines		File associated descrip lines
USERD_get_element_label_status		Element labels?
USERD_get_extra_gui_defaults	X	Gets the default values for the extra GUI members
USERD_get_extra_gui_numbers	X	Gets the number of toggles, pulldowns and fields
USERD_get_geom_timeset_number		Timeset # to use for geom
USERD_get_gold_part_build_info		Gets the info needed for part building process
USERD_get_gold_variable_info		Variable type/descrip etc
USERD_get_ghosts_in_model_flag		Model contains ghost cells?
USERD_get_matf_escalars_desc	X	Gets material scalars description (Youngs method)

USERD_get_matf_set_info		Gets the material set indices and names
USERD_get_matf_set_type	X	Gets the material set type
USERD_get_matf_var_info		Gets the material indices and descriptions
USERD_get_matsp_info		Gets material species id, descriptions, etc.
USERD_get_maxsize_info		Part/block allocation maximums
USERD_get_model_extents		Provide model bounding extents
USERD_get_name_of_reader		Name of reader for GUI
USERD_get_nfaced_conn		Gets the element connectivities for nfaced elements (utilizes the number of nodes per face obtained in USERD_get_nfaced_nodes_per_face)
USERD_get_nfaced_conn_in_buffers	X	Gets the element connectivities for nfaced elements in buffers. (For unstructured autodistrib for SOS)
USERD_get_nfaced_nodes_per_face		Gets the number of nodes per face for nfaced elements (utilizes the number of faces per element obtained in USERD_get_part_elements_by_type)
USERD_get_node_label_status		Node labels?
USERD_get_nsided_conn		Gets the element connectivities for nsided elements. (utilizes the number of nodes per element obtained in USERD_get_part_elements_by_type)
USERD_get_nsided_conn_in_buffers	X	Gets the element connectivities for nsided elements in buffers. (For unstructured autodistrib for SOS)
USERD_get_num_of_time_steps		Number of time steps
USERD_get_num_xy_queries	X	Number of xy queries
USERD_get_number_of_files_in_dataset		Number of files in model
USERD_get_number_of_material_sets		Number of material sets
USERD_get_number_of_materials		Number of materials
USERD_get_number_of_model_parts		Number of model parts
USERD_get_number_of_species		Number of species
USERD_get_number_of_timesets		Number of timesets
USERD_get_number_of_variables		Number of variables
USERD_get_part_element_ids_by_type		Part's element ids
USERD_get_part_element_ids_by_type_in_buffers	X	Part's element ids in buffers (For unstructured autodistrib for SOS)
USERD_get_part_node_ids		Part's node ids
USERD_get_part_node_ids_in_buffers	X	Part's node ids in buffers (For unstructured autodistrib for SOS)
USERD_get_periodic_ghosts_num_symmetry_faces	X	Returns the number of symmetry faces for each part
USERD_get_reader_descrip	X	Provide GUI more description
USERD_get_reader_release	X	Release string of reader
USERD_get_reader_version		Provide reader version number
USERD_get_sol_times		Solution time values

USERD_get_structured_reader_cinching		Tells if the reader will do structured cinching
USERD_get_timeset_description		Description of timeset
USERD_get_uns_failed_params		Gets variable and thresholds/criteria for failure
USERD_get_var_by_component		Part or block variable values
USERD_get_var_by_component_in_buffers	X	Part or block variable values in buffers (For unstructured autodistrib for SOS)
USERD_get_var_extract_gui_defaults	X	Gets the default values for the var_extract members
USERD_get_var_extract_gui_numbers	X	Gets the number of toggles, pulldowns and fields
USERD_get_var_value_at_specific		Node's or element's variable value over time
USERD_get_vglyph_counts	X	Gets counts for number of vector vglyphs
USERD_get_vglyph_timeline_info	X	Gets vector vglyph timeline metadata
USERD_get_vglyph_timeline_times	X	Gets vector vglyph timeline times
USERD_get_vglyph_vector_info	X	Gets vector vglyph metadata
USERD_get_vglyph_vector_values	X	Gets vector vglyph component values
USERD_get_vglyph_vector_xyzloc	X	Gets vector vglyph xyz locations
USERD_get_xy_query_data	X	Gets xy query xy values
USERD_get_xy_query_info	X	Gets xy query names, titles, num pairs, etc.
USERD_load_matf_data		Gets the material ids list, mixed-material ids list, or mixed-material values list
USERD_prefer_auto_distribute	X	Tells whether reader will distribute for SOS
USERD_rigidbody_existence		Returns whether rigid body transformation data exists for the model.
USERD_rigidbody_values		Returns the euler and location values for a given part
USERD_set_extra_gui_data	X	Returns the Extra GUI answers provided by the user
USERD_set_filename_button_labels	X	Sets Get File button text
USERD_set_filenames		Filenames entered in GUI
USERD_set_right_side	X	Informs the reader when the time set is for the right side of a time span during variable interpolation between time steps.
USERD_set_server_number		Server which of how many
USERD_set_time_set_and_step		Current timeset and time step
USERD_set_var_extract_gui_data	X	Returns the variable extract answers provided by the user
USERD_size_matf_data		Gets the length of either the material ids list, mixed-material ids list, or mixed-material values list
USERD_stop_part_building		Cleanup after part build routine
USERD_use_periodic_ghosts	X	Returns whether any parts have periodic symmetry

2.2 Order Routines are Called

It is often helpful in the development of your reader to know what order the routines will be called. The various main operations are given basically in the order they will be performed. Within each operation, the order the routines will be called is given.

1. Setting name in the gui, and specifying one or two input fields

   USERD_get_name_of_reader
   USERD_get_reader_descrip (optional)
   USERD_prefer_auto_distribute (optional)
   USERD_set_filename_button_labels (optional)
   USERD_get_extra_gui_numbers (optional)
   USERD_get_extra_gui_defaults (optional)
   USERD_get_reader_release (optional)

2. Getting the reader version (also distinguishes between API's)

   USERD_get_reader_version

3. Setting filenames and getting timeset and time info

   USERD_set_extra_gui_data (optional if reader has USERD_get_extra_gui_defaults routine)

   USERD_get_structured_reader_cinching
   USERD_set_server_number
   USERD_set_extra_gui_data (optional)
   USERD_set_filenames
   USERD_get_number_of_timesets
   USERD_get_geom_timeset_number

   for each timeset:

      USERD_get_timeset_description
      USERD_get_num_of_time_steps
      USERD_get_sol_times

   USERD_set_time_set_and_step

4. Gathering info for part builder

   USERD_set_time_set_and_step
   USERD_get_changing_geometry_status
   USERD_rigidbody_existence
   USERD_get_node_label_status
   USERD_get_element_label_status
   USERD_get_number_of_files_in_dataset
   USERD_get_dataset_query_file_info
   USERD_get_descrip_lines (for geometry)
   USERD_get_number_of_model_parts
   USERD_get_gold_part_build_info
   USERD_get_ghosts_in_model_flag
   USERD_get_maxsize_info
   USERD_get_ghosts_in_block_flag (if any ghost cells in model)
   USERD_get_model_extents -- or -- (for model extents)
      USERD_get_part_coords -- and/or --
      <USERD_set_block_range_and_stride> (if doing structured reader cinching)

USERD_get_block_coords_by_component
USERD_get_uns_failed_params

5. Gathering Variable info

USERD_get_number_of_variables
USERD_get_gold_variable_info

6. Part building (per part created)

Both unstructured and structured:

USERD_set_time_set_and_step

If unstructured part:

USERD_get_part_coords -- or --
USERD_get_part_coords_in_buffers (optional)
USERD_rigidbody_values (optional)
USERD_get_part_node_ids -- or --
USERD_get_part_node_ids_in_buffers (optional)
USERD_get_part_element_ids_by_type -- or --
USERD_get_part_element_ids_by_type_in_buffers (optional)
USERD_get_part_elements_by_type -- or --
USERD_get_part_elements_by_type_in_buffers (optional)

If any nsided elements:

USERD_get_nsided_conn -- or --
USERD_get_nsided_conn_in_buffers (optional)

If any nfaced elements:

USERD_get_nfaced_nodes_per_face
USERD_get_nfaced_conn -- or --
USERD_get_nfaced_conn_in_buffers (optional)

else if structured part:

USERD_get_block_iblanking
<USERD_set_block_range_and_stride> (If doing structured reader cinching)
USERD_get_block_coords_by_component
USERD_rigidbody_values (optional)
USERD_get_block_ghost_flags (If ghost cells in part)
USERD_get_part_node_ids (If node ids given)
USERD_get_part_element_ids_by_type (If element ids given)

both again:

USERD_get_border_availability (If border representation is selected)
USERD_get_border_elements_by_type (If border representation is selected)
USERD_stop_part_building (only once when part builder dialog is closed)

7. Loading Variables

constants:

USERD_set_time_set_and_step
USERD_get_constant_val

scalars/vectors/tensors:

USERD_get_descrip_lines
USERD_set_time_set_and_step
USERD_set_right_side (optional)
<USERD_set_block_range_and_stride> (If doing structured reader cinching)
USERD_get_var_by_component -- or --
USERD_get_var_by_component_in_buffers (optional)

8. Changing geometry

changing coords only (per part):

USERD_set_time_set_and_step
USERD_get_descrip_lines
USERD_get_part_coords -- or --
USERD_get_part_coords_in_buffers (optional)
<USERD_set_block_range_and_stride> (If doing structured reader cinching)
USERD_get_block_coords_by_component

changing connectivity (per part):

Both unstructured and structured:

USERD_set_time_set_and_step
USERD_get_descrip_lines
USERD_get_number_of_model_parts
USERD_get_gold_part_build_info
USERD_get_ghosts_in_model_flag

If unstructured part:

USERD_get_model_extents -- or --
USERD_get_part_coords
USERD_get_part_coords -- or --
USERD_get_part_coords_in_buffers (optional)
USERD_rigidbody_values (optional)
USERD_get_part_node_ids -- or --
USERD_get_part_node_ids_in_buffers (optional)
USERD_get_part_element_ids_by_type -- or --
USERD_get_part_element_ids_by_type_in_buffers (optional)
USERD_get_part_elements_by_type -- or --
USERD_get_part_elements_by_type_in_buffers (optional)

If any nsided elements:

USERD_get_nsided_conn -- or --
USERD_get_nsided_conn_in_buffers (optional)

If any nfaced elements:

USERD_get_nfaced_nodes_per_face
USERD_get_nfaced_conn -- or --
USERD_get_nfaced_conn_in_buffers (optional)

else if structured part:

USERD_get_model_extents -- or --
USERD_get_part_coords
USERD_get_block_iblanking
<USERD_set_block_range_and_stride> (If doing structured reader cinching)

USERD_get_block_coords_by_component
USERD_rigidbody_values (optional)
USERD_get_block_ghost_flags (If ghost cells in part)
USERD_get_part_node_ids (If node ids given)
USERD_get_part_element_ids_by_type (If element ids given)

both again:

USERD_get_border_availability (If border representation is selected)
USERD_get_border_elements_by_type (If border representation is selected)

9. Node or Element queries over time

   USERD_get_var_value_at_specific

10. To see if materials in the model

    USERD_get_number_of_material_sets
    USERD_get_matf_set_info

    If any material sets in the model (calls these once per material set):

    USERD_get_number_of_materials
    USERD_get_matf_var_info
    USERD_get_matf_set_type
    USERD_get_matf_escalars_desc

    For each element type of each part containing material ids, calls:

    USERD_size_matf_data
    USERD_load_matf_data

    If there are any elements with mixed materials, when a domain or interface is created, calls these again per part:

    USERD_size_matf_data
    USERD_load_matf_data

11. To modify the variable extraction parameters and have the variables update accordingly.

    USERD_get_var_extract_gui_numbers
    USERD_get_var_extract_gui_defaults
    USERD_set_var_extract_gui_data

12. To get vector vglyphs.

    USERD_get_vglyph_counts
    USERD_get_vglyph_timeline_info
    USERD_get_vglyph_timeline_times
    USERD_get_vglyph_vector_info
    USERD_get_vglyph_vector_values
    USERD_get_vglyph_vector_xyzloc

2.3 Routine History

The following table is an alphabetical listing of the routines in the API. It indicates at which version the routines appeared (or were modified). Additionally it indicates which routines are optional (OPT).

Routine Name	OPT	2.00	2.01	2.03	2.04	2.05	2.06	2.07	2.08	2.09	2.10
USERD_bkup		X	X	X	X	X	X	X	X	X	X
USERD_exit_routine		X	X	X	X	X	X	X	X	X	X
USERD_get_block_coords_by_component		X	X	X	X	X	X	X	X	X	X
USERD_get_block_iblanking		X	X	X	X	X	X	X	X	X	X
USERD_get_block_ghost_flags			X	X	X	X	X	X	X	X	X
USERD_get_border_availability		X	X	X	X	X	X	X	X	X	X
USERD_get_border_elements_by_type		X	X	X	X	X	X	X	X	X	X
USERD_get_changing_geometry_status		X	X	X	X	X	X	X	X	X	X
USERD_get_constant_val		X	X	X	X	X	X	X	X	X	X
USERD_get_dataset_query_file_info		X	X	X	X	X	X	X	X	X	X
USERD_get_descrip_lines		X	X	X	X	X	X	X	X	X	X
USERD_get_element_label_status		X	1	X	X	X	X	X	X	X	X
USERD_get_extra_gui_defaults	X	X	X	X	X	X	X	X	X	X	X
USERD_get_extra_gui_numbers	X	X	X	X	X	X	X	X	X	X	X
USERD_get_geom_timeset_number		X	X	X	X	X	X	X	X	X	X
USERD_get_gold_part_build_info		X	2	3	X	X	X	X	X	X	X
USERD_get_gold_variable_info		X	X	X	X	X	X	X	X	X	X
USERD_get_ghosts_in_block_flag			X	X	X	X	X	X	X	X	X
USERD_get_ghosts_in_model_flag			X	X	X	X	X	X	X	X	X
USERD_get_matf_escalars_desc	X										X
USERD_get_matf_set_info				X	X	X	X	X	X	X	X
USERD_get_matf_set_type	X										X
USERD_get_matf_var_info				X	X	X	X	X	X	X	X
USERD_get_matsp_info						X	X	X	X	X	X
USERD_get_maxsize_info		X	2	X	X	X	X	X	X	X	X
USERD_get_model_extents		X	X	X	X	X	X	X	X	X	X

USERD_get_name_of_reader		X	X	X	X	X	X	X	X	X	X
USERD_get_nfaced_conn				X	X	X	X	X	X	X	X
USERD_get_nfaced_conn_in_buffers	X								X	X	X
USERD_get_nfaced_nodes_per_face				X	X	X	X	X	X	X	X
USERD_get_node_label_status		X	1	X	X	X	X	X	X	X	X
USERD_get_nsided_conn				X	X	X	X	X	X	X	X
USERD_get_nsided_conn_in_buffers	X								X	X	X
USERD_get_num_of_time_steps		X	X	X	X	X	X	X	X	X	X
USERD_get_num_xy_queries	X								X	X	X
USERD_get_number_of_files_in_dataset		X	X	X	X	X	X	X	X	X	X
USERD_get_number_of_material_sets				X	X	X	X	X	X	X	X
USERD_get_number_of_materials				X	X	X	X	X	X	X	X
USERD_get_number_of_model_parts		X	X	X	X	X	X	X	X	X	X
USERD_get_number_of_species						X	X	X	X	X	X
USERD_get_number_of_timesets		X	X	X	X	X	X	X	X	X	X
USERD_get_number_of_variables		X	X	X	X	X	X	X	X	X	X
USERD_get_part_coords		X	X	X	X	X	X	X	X	X	X
USERD_get_part_coords_in_buffers	X								X	X	X
USERD_get_part_element_ids_by_type		X	1	2	X	X	X	X	X	X	X
USERD_get_part_element_ids_by_type_in_buffers	X								X	X	X
USERD_get_part_elements_by_type		X	1	2	X	X	X	X	X	X	X
USERD_get_part_elements_by_type_in_buffers	X								X	X	X
USERD_get_part_node_ids		X	1	2	X	X	X	X	X	X	X
USERD_get_part_node_ids_in_buffers	X								X	X	X
USERD_get_reader_descrip	X	X	X	X	X	X	X	X	X	X	X
USERD_get_reader_release	X	X	X	X	X	X	X	X	X	X	X
USERD_get_reader_version		X	X	X	X	X	X	X	X	X	X
USERD_get_sol_times		X	X	X	X	X	X	X	X	X	X
USERD_get_structured_reader_cinching							X	X	X	X	X
USERD_get_timeset_description		X	X	X	X	X	X	X	X	X	X
USERD_get_uns_failed_params					X	X	X	X	X	X	X
USERD_get_var_by_component		X	2	X	X	X	X	X	X	X	X
USERD_get_var_by_component_in_buffers	X								X	X	X

USERD_get_var_extract_gui_defaults	X					X	X	X	X	X	X
USERD_get_var_extract_gui_numbers	X					X	X	X	X	X	X
USERD_get_var_value_at_specific		X	X	X	X	X	X	X	X	X	X
USERD_get_vglyph_counts	X									X	X
USERD_get_vglyph_timeline_info	X									X	X
USERD_get_vglyph_timeline_times	X									X	X
USERD_get_vglyph_vector_info	X									X	X
USERD_get_vglyph_vector_values	X									X	X
USERD_get_vglyph_vector_xyzloc	X									X	X
USERD_get_xy_query_data	X								X	X	X
USERD_get_xy_query_info	X								X	X	X
USERD_load_matf_data				X	X	X	X	X	X	X	X
USERD_prefer_auto_distribute	X							X	X	X	X
USERD_rigidbody_existence						X	X	X	X	X	X
USERD_rigidbody_values						X	X	X	4	X	X
USERD_set_block_range_and_stride							X	X	X	X	X
USERD_set_extra_gui_data	X	X	X	X	X	X	X	X	X	X	X
USERD_set_filename_button_labels	X							X	X	X	X
USERD_set_filenames		X	X	X	X	X	X	X	X	X	X
USERD_set_right_side	X					X	X	X	X	X	X
USERD_set_server_number		X	X	X	X	X	X	X	X	X	X
USERD_set_time_set_and_step		X	X	X	X	X	X	X	X	X	X
USERD_set_var_extract_gui_data	X					X	X	X	X	X	X
USERD_size_matf_data				X	X	X	X	X	X	X	X
USERD_stop_part_building		X	X	X	X	X	X	X	X	X	X

Footnotes:

1 Modifications due to user specified ids for structured blocks

2 Additional valid ghost element types available

3 Modifications for specification of structured ranges

4 Added yaw, pitch, roll

At Version 2.00

These routines existed in the original 2.00 version.

USERD_bkup
USERD_exit_routine
USERD_get_block_coords_by_component
USERD_get_block_iblanking
USERD_get_border_availability
USERD_get_border_elements_by_type
USERD_get_changing_geometry_status
USERD_get_constant_val
USERD_get_dataset_query_file_info
USERD_get_descrip_lines
USERD_get_element_label_status
USERD_get_extra_gui_defaults <optional>
USERD_get_extra_gui_numbers <optional>
USERD_get_geom_timeset_number
USERD_get_gold_part_build_info
USERD_get_gold_variable_info
USERD_get_maxsize_info
USERD_get_model_extents
USERD_get_name_of_reader
USERD_get_node_label_status
USERD_get_num_of_time_steps
USERD_get_number_of_files_in_dataset
USERD_get_number_of_model_parts
USERD_get_number_of_timesets
USERD_get_number_of_variables
USERD_get_part_coords
USERD_get_part_element_ids_by_type
USERD_get_part_elements_by_type
USERD_get_part_node_ids
USERD_get_reader_descrip <optional>
USERD_get_reader_release <optional>
USERD_get_reader_version
USERD_get_sol_times
USERD_get_timeset_description
USERD_get_var_by_component
USERD_get_var_value_at_specific
USERD_set_extra_gui_data <optional>
USERD_set_filenames
USERD_set_server_number
USERD_set_time_set_and_step
USERD_stop_part_building

At Version 2.01

ADDED for Ghost Cell support:

USERD_get_block_ghost_flags
USERD_get_ghosts_in_block_flag
USERD_get_ghosts_in_model_flag

MODIFIED for user specified ids for structured blocks:

USERD_get_element_label_status
USERD_get_node_label_status
USERD_get_part_elements_by_type
USERD_get_part_node_ids

MODIFIED for Ghost Cell support:

USERD_get_gold_part_build_info
USERD_get_maxsize_info
USERD_get_part_element_ids_by_type
USERD_get_part_elements_by_type
USERD_get_var_by_component

At Version 2.03

ADDED to handle material sets:

USERD_get_matf_set_info
USERD_get_matf_var_info
USERD_get_number_of_material_sets
USERD_get_number_of_materials
USERD_load_matf_data
USERD_size_matf_data

ADDED to handle nsided and nfaced elements:

USERD_get_nfaced_conn
USERD_get_nfaced_nodes_per_face
USERD_get_nsided_conn

MODIFIED so structured ranges can be specified:

USERD_get_gold_part_build_info

At Version 2.04

ADDED to handle failed elements:

(Can implement to specify to EnSight, which variable is the failed element variable and what the conditions of failure are)

USERD_get_uns_failed_params

At Version 2.05

ADDED to handle material species:

USERD_get_matsp_info
USERD_get_number_of_species

ADDED to handle variable extraction after a read:

(This is similar to Extra GUI options, but will modify the variable extraction options after the initial read - and update the variables accordingly)

USERD_get_var_extract_gui_defaults <optional>
USERD_get_var_extract_gui_numbers <optional>
USERD_set_var_extract_gui_data <optional>

ADDED to obtain rigid body values:

(If you can provide euler parameters for rigid body motion of parts, you should implement these routines)

USERD_rigidbody_existence
USERD_rigidbody_values

ADDED to let reader know when on right side of a time interval for var values:

(When the current time is between 2 given time steps - requiring interpolation of variable values - EnSight asks for the left then the right side values. For most readers, you never need to know this. But, if you must do some efficient interpolation within the reader itself because of differing timelines - this can be useful information)

USERD_set_right_side <optional>

At Version 2.06

ADDED to allow structured readers to deal with min, max, and stride in reader:

(To keep from having to send the entire non-strided block to EnSight - and having it then limit the processing - you can implement this routine, and deal with the limiting and striding within the reader itself. Allowing for lower memory requirements. ***If you want to use structured auto-distribute, you must implement these routines.***)

USERD_get_structured_reader_cinching
USERD_set_block_range_and_stride

At Version 2.07

ADDED to allow specification of whether the reader will auto distribute within itself for SOS:

(If your reader can (and will) partition based on which server of the total number of severs - you will want to provide this routine)

USERD_prefer_auto_distribute <optional>

ADDED to allow readers to specify their own label for Set button in EnSight:

USERD_set_filename_button_labels <optional>

At Version 2.08

ADDED for efficient unstructured autodistribute capability:

(If you want to be able to use the unstructured autodistribute capability for SOS processing, in EnSight 8.2 or later, you should implement these routines):

Note: These five routines are for normal elements. *If any of them are implemented, they all must be implemented.*

USERD_get_part_coords_in_buffers <optional>
USERD_get_part_element_ids_by_type_in_buffers <optional>
USERD_get_part_elements_by_type_in_buffers <optional>
USERD_get_part_node_ids_in_buffers <optional>
USERD_get_var_by_component_in_buffers <optional>

This one routine for nsided elements.

USERD_get_nsided_conn_in_buffers <optional>

This one optional routine for nfaced elements

USERD_get_nfaced_conn_in_buffers <optional>

Unstructured Auto Distribute is a capability requiring Server of Servers (SOS) that will partition an unstructured model for you automatically across a set of servers.

If you do not implement the routines listed above (and described below) in your reader, EnSight can still perform this operation but will require much more memory on each server to read in the data (somewhat like each server having to read the whole model). You will however, get the execution advantage of having your model partitioned across multiple servers.

If you do implement these routines in your reader (in a proper manner), you should be able to not only get the execution advantages, but also memory usage on each server which is proportional to the subset that it is assigned to deal with.

Note that the optional routines are functionally quite similar to the following functions. And thus their implementation should not be too difficult to add to any existing reader that has already implemented these:

USERD_get_part_coords
USERD_get_part_node_ids
USERD_get_part_elements_by_type
USERD_get_part_element_ids_by_type
USERD_get_var_by_component
USERD_get_nsided_conn
USERD_get_nfaced_conn

ADDED for providing xy plot data out of a reader:

(If your data format provides plot/query xy data, you can implement these routines to have that data be available to EnSight's plotter)

USERD_get_num_xy_queries <optional>
USERD_get_xy_query_data <optional>
USERD_get_xy_query_info <optional>

MODIFIED for allowing yaw, pitch, roll:

(In addition to the specification of euler parameters and translations, and initial translational offsets, this routine was modified to allow for initial yaw, pitch, roll transformations as well.)

USERD_rigidbody_values

At Version 2.09

ADDED for providing vector glyph data out of a reader:

(If your reader can (and will) provide vector glyph data (generally used for annotating force and/or moment vectors), you will want to provide the following routines)

USERD_get_vglyph_counts <optional>
USERD_get_vglyph_timeline_info <optional>
USERD_get_vglyph_timeline_times <optional>
USERD_get_vglyph_vector_info <optional>
USERD_get_vglyph_vector_values <optional>
USERD_get_vglyph_vector_xyzloc <optional>

At Version 2.10

ADDED for allowing materials to be specified as a scalar:

(If your reader can (and will) provide material data as per element scalars, you will want to provide the following routines)

USERD_get_matf_set_type <optional>
USERD_get_matf_escalars_desc <optional>

2.4 Detailed Specifications

Include files:

The following header file is required in any file containing these library routines.

```
#include "global_extern.h"
```

And it references:

```
#include "global_extern_proto.h"
```

Special Note:

Make sure you use the proper define in the global_extern.h header file, namely:

#define USERD_API_208

Also, make sure the api version in the USERD_get_reader_version routine is set to the desired version.

Basis of arrays:

Unless explicitly stated otherwise, all arrays are zero based - in true C fashion.

Global variables:

You will generally need to have a few global variables which are shared by the various library routines. The detailed specifications below have assumed the following are available. (Their names describe their purpose, and they will be used in helping describe the details of the routines below).

```
static int Numparts_available        = 0;
static int Num_unstructured_parts    = 0;
static int Num_structured_blocks     = 0;

/* Note: Numparts_available = Num_unstructured_parts + Num_structured_blocks */

static int Num_timesets              = 1;
static int Current_timeset           = 1;
static int Geom_timeset_number       = 1;

static int Num_time_steps[Z_MAXSETS] = 1;
static int Current_time_step         = 0;
static int Num_variables             = 0;
static int Num_dataset_files         = 0;

static int Server_Number             = 1;
static int Tot_Servers               = 1;
```

Dummy (or stub) Routines:

Those routines marked optional, need not be included in a reader. They are truly optional. All other routines for a given version number need to be included, but can often be dummy routines - depending on what is returned for other related routines. As an example, if you always return that borders are not available in USERD_get_border_availability, then the USERD_get_border_elements_by_type routine can be a dummy routine - because it will never be called.

The specifications for each routine in the API will now be given (routines are in alphabetical order):

```
/*-----------------------------------------------------------------------
 USERD_bkup
 *                                          (version 2.00 and later)
 *-----------------------------------------------------------------------
 *
 *   Used in the archive process.  Save or restore info relating to
 *   your user defined reader.
 *
 *  (IN)  archive_file         = The archive file pointer
 *
 *  (IN)  backup_type          = Z_SAVE_ARCHIVE for saving archive
 *                               Z_REST_ARCHIVE for restoring archive
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Since EnSight's archive file is saved in binary form, it is
 *    suggested that you also do any writing to it or reading from it
 *    in binary.
 *
 *  * You should archive any variables, that will be needed for
 *    future operations, that will not be read or computed again
 *    before they will be needed.  These are typically global
 *    variables.
 *
 *  * Make sure that the number of bytes that you write on a save and
 *    the number of bytes that you read on a restore are identical!!
 *
 *  * And one last reminder.  If any of the variables you save are
 *    allocated arrays, you must do the allocations before restoring
 *    into them.
 *
 *  ================================
 *  * SPECIAL NOTE FOR WINDOWS ONLY:
 *    Because our current implementation under windows needs to open and close files
 *    from within the reader .dll, a special structure (named USERD_globals) needs to
 *    be defined in the global space of your reader. This structure needs to be defined
 *    like:              ------ -----
 *
 *    #ifdef WIN32                          (which includes 32 bit and 64 bit windows)
 *    W32EXPORT struct _USERD_globals {
 *      char arch_filename[256];
 *      unsigned long arch_fileptr;
 *    } USERD_globals;
 *    #endif
 *
 *    This structure will be bound when the reader .dll is loaded and will be used to
 *    store the archive file name and the current offset therein.
 *    Again for windows only, you need to ignore the archive_file pointer in the
 *    argument list and instead open and close the arch_filename file as well as keep
 *    the arch_fileptr offset current in this routine.
 *
 *    So first define the USERD_globals structure at the beginning of your reader.
 *
 *    Then, when an archive is saved, the following needs to be done in this routine:
 *      1. open USERD_globals.arch_filename for appending         (within #ifdef WIN32)
 *      2. do your writes
 *      3. close the file                                         (within #ifdef WIN32)
 *
 *    When an archive is restored, do the following in this routine:
```

```
 *      1. open USERD_globals.arch_filename for reading,
 *            and fseek to USERD_globals.arch_fileptr offset      (within #ifdef WIN32)
 *      2. do your reads
 *      3. save the new USERD_globals.arch_fileptr offset (using ftell),
 *            and close the file                                  (within #ifdef WIN32)
 *
 *    Here is some pseudo code to illustrate:
 *    ----------------------------------------
 *    switch(baskup_type) {
 *    case Z_SAVE_ARCHIVE:
 *
 *  #ifdef WIN32
 *      archive_file = fopen(USERD_globals.arch_filename,"ab");
 *  #endif
 *
 *       .
 *       .
 *       .
 *  #ifdef WIN32
 *      fclose(archive_file)
 *  #endif
 *
 *      break;
 *
 *    case Z_REST_ARCHIVE:
 *
 *  #ifdef WIN32
 *      archive_file = fopen(USERD_globals.arch_filename,"rb");
 *      fseek(archive_file, USERD_globals.arch_fileptr, SEEK_SET);
 *  #endif
 *
 *       .
 *       .
 *       .
 *
 *  #ifdef WIN32
 *      USERD_globals.arch_fileptr = ftell(archive_file);
 *      fclose(archive_file)
 *  #endif
 *
 *      break;
 *   }
 *
 *  And finally be aware of a current limitation of the
 *  Windows implementation of this routine:
 *  ---------------------------------------
 *  Because the structure uses a long for the file offset, the archive restore
 *  will not work when the offset to the information written in this routine
 *  is greater than 2 Gb, on 32 bit windows. On 64 bit windows there is no such
 *  limitation because the long is 64 bits.
 *-----------------------------------------------------------------------*/
int
USERD_bkup(FILE *archive_file,
           int backup_type)
```

```
/*----------------------------------------------------------------------
 USERD_exit_routine
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Called when EnSight is exited for USERD, can be used to
 *   clean up temporary files, etc. It is often simply a dummy.
 *----------------------------------------------------------------------*/
void
USERD_exit_routine( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_block_coords_by_component
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the coordinates of a given block, component at a time
 *
 *  (IN)  block_number            = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  which_component         = Z_COMPX if x component wanted
 *                                = Z_COMPY if y component wanted
 *                                = Z_COMPZ if z component wanted
 *
 *  (OUT) coord_array             = 1D array containing x,y, or z
 *                                   coordinate component of each node
 *
 *                                  (Array will have been allocated
 *                                   i*j*k for the block long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Only called for structured "block" parts
 *----------------------------------------------------------------------*/
int
USERD_get_block_coords_by_component(int block_number,
                                    int which_component,
                                    float *coord_array)
```

```
/*----------------------------------------------------------------------
 *
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the iblanking value at each node of a block - If Z_IBLANKED
 *
 *  (IN)  block_number            = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) iblank_array            = 1D array containing iblank value
 *                                   for each node.
 *
 *                                  (Array will have been allocated
 *                                   i*j*k for the block long)
 *
 *          possible values are:   Z_EXT     = exterior (outside)
 *                                 Z_INT     = interior (inside)
 *                                 Z_BND     = boundary
 *                                 Z_INTBND  = internal boundary
 *                                 Z_SYM     = symmetry plane
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_structured_blocks is > 0  and you have
 *      some iblanked blocks
 *
 *  * Only called for structured "block" parts
 *----------------------------------------------------------------------*/
int
USERD_get_block_iblanking(int block_number,
                          int *iblank_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_block_ghost_flags
 *                                              (version 2.01 and later)
 *----------------------------------------------------------------------
 *
 *   Get the ghost_flags value at each element of a block containg ghost cells.
 *
 *  (IN)  block_number            = The block number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                  It is NOT the part_id that is
 *                                  loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) ghost_flags             = 1D array containing ghost flag value
 *                                   for each block cell.
 *
 *                                  (Array will have been allocated
 *                                   (i-1)*(j-1)*(k-1) for the block long)
 *
 *          possible values are:    0  = non-ghost cell  (normal cell)
 *                                 >0  = ghost cell
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Only called for structured "block" parts that have some ghost cells
 *    as indicated by the USERD_get_ghost_in_block_flag.  The model must
 *    of course also have been indicated to have some ghost cells in the
 *    USERD_get_ghost_in_model_flag routine.
 *
 *  * It is sufficient to set the value to be 1 to flag as a ghost cell,
 *    but the value can be any non-zero value, so you could use it to
 *    indicate which block or which server (for Server-of-server use) the
 *    cell is actually in.
 *----------------------------------------------------------------------*/
int
USERD_get_block_ghost_flags(int block_number,
                            int *ghost_flags)
```

```
/*----------------------------------------------------------------------
 USERD_get_border_availability
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Finds out if border elements are provided by the reader for the
 *   desired part, or will need to be computed internally by EnSight.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *   (OUT) number_of_elements     = 2D array containing number of
 *                                  each type of border element in
 *                                  the part.
 *                                  ------------
 *                                  Possible types are:
 *
 *                                  Z_POINT   =  point
 *                                  Z_BAR02   =  2-noded bar
 *                                  Z_BAR03   =  3-noded bar
 *                                  Z_TRI03   =  3-noded triangle
 *                                  Z_TRI06   =  6-noded triangle
 *                                  Z_QUA04   =  4-noded quadrilateral
 *                                  Z_QUA08   =  8-noded quadrilateral
 *
 *  Returns:
 *  -------
 *  Z_OK  if border elements will be provided by the reader.
 *         (number_of_elements array will be loaded and
 *          USERD_get_border_elements_by_type will be called)
 *
 *  Z_ERR if border elements are not available - thus EnSight must compute.
 *         (USERD_get_border_elements_by_type will not be called)
 *
 *
 *  Notes:
 *  -----
 *  * Only called if border representation is used.
 *
 *  * Will be based on Current_time_step
 *
 *----------------------------------------------------------------------*/
int
USERD_get_border_availability( int part_number,
                               int number_of_elements[Z_MAXTYPE])
```

```
/*----------------------------------------------------------------------
 USERD_get_border_elements_by_type
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Provides border element connectivity and parent information.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  element_type            = One of the following (See global_extern.h)
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *
 *  (OUT) conn_array              = 2D array containing connectivity
 *                                  of each border element of the type.
 *
 *                                 (Array will have been allocated
 *                                  num_of_elements of the type by
 *                                  connectivity length of the type)
 *
 *                      ex) If number_of_elements[Z_TRI03] = 25
 *                             number_of_elements[Z_QUA04] = 100
 *                             number_of_elements[Z_QUA08] = 30
 *                          as obtained in:
 *                           USERD_get_border_availability
 *
 *                          Then the allocated dimensions available
 *                          for this routine will be:
 *                             conn_array[25][3]   when called with Z_TRI03
 *
 *                             conn_array[100][4]  when called with Z_QUA04
 *
 *                             conn_array[30][8]   when called with Z_QUA08
 *
 *  (OUT) parent_element_type   = 1D array containing element type of the
 *                                parent element (the one that the border
 *                                element is a face/edge of).
 *
 *                               (Array will have been allocated
 *                                num_of_elements of the type long)
 *
 *  (OUT) parent_element_num    = 1D array containing element number of the
 *                                parent element (the one that the border
 *                                element is a face/edge of).
 *
 *                               (Array will have been allocated
 *                                num_of_elements of the type long)
 *
 *
```

```
 *  Returns:
 *  -------
 *  Z_OK  if successful
 *  Z_ERR if not successful
 *
 *  Notes:
 *  -----
 *  * Only called if USERD_get_border_availability returned Z_OK
 *
 *  * Will be based on Current_time_step
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_border_elements_by_type( int part_number,
                                   int element_type,
                                   int **conn_array,
                                   short *parent_element_type,
                                   int *parent_element_num)
```

```
/*----------------------------------------------------------------------
 USERD_get_changing_geometry_status
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the changing geometry status
 *
 *  returns:  Z_STATIC        if geometry does not change
 *            Z_CHANGE_COORDS if changing coordinates only
 *            Z_CHANGE_CONN   if changing connectivity
 *
 *  Notes:
 *  * EnSight does not support changing number of parts.  But the
 *    coords and/or the connectivity of the parts can change. Note that
 *    a part is allowed to be empty (number of nodes and elements equal
 *    to zero).
 *----------------------------------------------------------------------*/
int
USERD_get_changing_geometry_status( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_constant_val
 *                                            (version 2.00 and later)
 *---------------------------------------------------------------------
 *
 *   Get the value of a constant at a time step
 *
 *  (IN)  which_var            = The variable number (1 to Num_variables)
 *
 *  (IN)  imag_data            = TRUE if want imaginary data value.
 *                               FALSE if want real data value.
 *
 *  returns: value of the requested constant variable
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *---------------------------------------------------------------------*/
float
USERD_get_constant_val(int which_var,
                       int imag_data)
```

```
/*----------------------------------------------------------------------
 USERD_get_dataset_query_file_info
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the information about files in the dataset.  Used for the
 *   dataset query option within EnSight.
 *
 *  (OUT) qfiles   = Structure containing information about each file
 *                   of the dataset. The Z_QFILES structure is defined
 *                   in the global_extern.h file
 *
 *                   (The structure will have been allocated
 *                    num_dataset_files long, with 10 description
 *                    lines per file).
 *                    (See USERD_get_number_of_files_in_dataset)
 *
 *      qfiles[].name        = The name of the file
 *                             (Z_MAXFILENP is the dimensioned length
 *                              of the name)
 *
 *      qfiles[].sizeb       = The number of bytes in the file
 *                             (Typically obtained with a call to the
 *                              "stat" system routine)
 *
 *      qfiles[].timemod     = The time the file was last modified
 *                             (Z_MAXTIMLEN is the dimesioned length
 *                              of this string)
 *                             (Typically obtained with a call to the
 *                              "stat" system routine)
 *
 *      qfiles[].num_d_lines = The number of description lines you
 *                              are providing from the file. Max = 10
 *
 *      qfiles[].f_desc[]    = The description line(s) per file,
 *                              qfiles[].num_d_lines of them
 *                              (Z_MAXFILENP is the allocated length of
 *                               each line)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * If num_dataset_files is 0, this routine will not be called.
 *    (See USERD_get_number_of_files_in_dataset)
 *----------------------------------------------------------------------*/
int
USERD_get_dataset_query_file_info(Z_QFILES *qfiles)
```

```
/*----------------------------------------------------------------------
 *                                                  (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get two description lines associated with geometry per time step,
 *   or one description line associated with a variable per time step.
 *
 *  (IN)  which_type           = Z_GEOM for geometry
 *                             = Z_VARI for variable
 *
 *  (IN)  which_var            = If it is a variable, which one. (1 to Num_variables)
 *                               Ignored if geometry type.
 *
 *  (IN)  imag_data            = TRUE if want imaginary data file.
 *                               FALSE if want real data file.
 *
 *  (OUT) line1                = The 1st geometry description line,
 *                               or the variable description line.
 *
 *  (OUT) line2                = The 2nd geometry description line
 *                               Not used if variable type.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * These are the lines EnSight can echo to the screen in
 *    annotation mode.
 *----------------------------------------------------------------------*/
int
USERD_get_descrip_lines(int which_type,
                        int which_var,
                        int imag_data,
                        char line1[Z_BUFL],
                        char line2[Z_BUFL])
```

USERD_get_descrip_lines

```
/*----------------------------------------------------------------------
 USERD_get_element_label_status
 *                                              (version 2.00 and later)
 *                                 (Modified at 2.01 as indicated below)
 *----------------------------------------------------------------------
 *
 *  Answers the question as to whether element labels will be provided.
 *
 *  returns:  TRUE        if element labels will be provided
 *            FALSE       if element labels will NOT be provided
 *
 *  Notes:
 *  * These are needed in order to do any element querying, or
 *    element labeling on-screen within EnSight.
 *
 *  * Will call USERD_get_part_element_ids_by_type for each type of
 *    of each part if this routine returns TRUE.
 *
 *  * Prior to API 2.01:
 *    =================
 *      For unstructured parts, you can read them from your file if
 *      available, or can assign them, etc. They need to be unique
 *      per part, and are often unique per model.
 *
 *      API 1.0:
 *        USERD_get_element_ids_for_part is used to obtain the ids,
 *        on a part by part basis, if TRUE status is returned here.
 *
 *      API 2.0:
 *        USERD_get_part_element_ids_by_type is used to obtain the ids,
 *        on an element type by part basis, if TRUE status is returned here.
 *
 *      For structured parts, EnSight will assign ids if you return a
 *        status of TRUE here.  You cannot assign them youself!!
 *
 *   * Starting at API 2.01:
 *     ====================
 *      For both unstructured and structured parts, you can read them
 *      from your file if available, or can assign them, etc. They need
 *      to be unique per part, and are often unique per model (especially
 *      if you are dealing with a decomposed dataset).
 *
 *      USERD_get_part_element_ids_by_type is used to obtain the ids,
 *      on an element type by part basis, if TRUE status is returned here.
 *----------------------------------------------------------------------*/
int
USERD_get_element_label_status( void )
```

```
/*-----------------------------------------------------------------------
 USERD_get_extra_gui_defaults
 *                                      <optional> (version 2.00 and later)
 *-----------------------------------------------------------------------
 *
 *  This routine defines the Titles, status, List choices, strings, etc that
 *  are fed up to the GUI.
 *
 *   (OUT) toggle_Title               = title for each toggle
 *                                      array dimension is
 *                                  [num_toggles] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) toggle_default_status      = Setting for each toggle (TRUE or FALSE)
 *                                      array dimension is [num_toggles] long
 *
 *   (OUT) pulldown_Title             = title for each pulldown
 *                                      array dimension is
 *                                  [num_pulldowns] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) pulldown_number_in_list    = number of items in each pulldown
 *                                      array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_default_selection = item selection for each pulldown
 *                                      array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_item_strings      = pulldown item strings
 *                                      array is [num_pulldowns] by
 *                                             [Z_MAX_NUM_GUI_PULL_ITEMS] by
 *                                             [Z_LEN_GUI_PULL_STR] long
 *
 *   (OUT) field_Title                = title for each field
 *                                      array dimension is
 *                                   [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) field_user_string          = content of the field
 *                                      array dimension is
 *                                   [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The library is loaded, this routine is called,
 *    then the library is unloaded.
 *
 *  * Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 * ---------------------------------------------- */
int USERD_get_extra_gui_defaults(char **toggle_Title,
                                 int *toggle_default_status,
                                 char **pulldown_Title,
                                 int *pulldown_number_in_list,
                                 int *pulldown_default_selection,
                                 char ***pulldown_item_strings,
                                 char **field_Title,
                                 char **field_user_string)
```

```
/*-----------------------------------------------------------------------
 USERD_get_extra_gui_numbers
 *                                <optional> (version 2.00 and later)
 * ----------------------------------------------------------------------
 *
 *   The Enhanced GUI routines are added to allow the user to customize a
 *   portion of the Data Reader dialog to pass in options to their user
 *   defined reader.
 *
 *   This routine defines the numbers of toggles, pulldowns & fields
 *
 *   (OUT) num_Toggles    = number of toggles that will be provided
 *
 *   (OUT) num_pulldowns  = number of pulldowns that will be provided
 *
 *   (OUT) num_fields     = number of fields that will be provided
 *
 *   Notes:
 *   * There are three routines that work together:
 *         USERD_get_extra_gui_numbers
 *         USERD_get_extra_gui_defaults (this one)
 *         USERD_set_extra_gui_data
 *
 *     The existence of these routine indicates that
 *     you wish to add customize entries to the
 *     Data Reader dialog.
 *
 *     If you don't want the extra GUI features,
 *     simply delete these routines, or change their
 *     names to something such as
 *     USERD_DISABLED_get_extra_gui_defaults
 *
 *     The presence of these routines
 *     will ensure that EnSight will call them and
 *     use their data to modify the Data Reader dialog
 *     with some or all of the following:
 *     toggles, pulldown menu and fields.
 *
 *     The user can then interact with the enhanced
 *     GUI and then send their choices to
 *     USERD_set_extra_gui_data
 *
 *     Therefore if USERD_get_extra_gui_numbers
 *     exists then the other two must exist.
 *
 *     If none exist, then the GUI will be unchanged.
 *
 *     Toggle data will return an integer
 *                               TRUE if checked
 *                               FALSE if unchecked
 *
 *     Pulldown menu will return an integer representing
 *                               the menu item selected
 *
 *     Field will return a string Z_LEN_GUI_FIELD_STR long.
 *
 *     If all the enhanced GUI features are enabled it
 *     might look something like this
 *
 *
 *
```

```
 *     ======================================================
 *
 *        [ ] Title 1
 *        [X] Title 3
 *        [X] Title 2
 *        [X] Title 4
 *
 *        Pulldown Menu ->
 *            Menu Choice 1
 *            Menu Choice 2
 *            Menu Choice 3
 *
 *        Data Field Title 1 _________________________________
 *
 *        Data Field Title 2 _________________________________
 *
 *     ======================================================
 *
 * * The following are defined in the global_extern.h
 *          Z_MAX_NUM_GUI_PULL_ITEMS max num GUI pulldowns
 *          Z_LEN_GUI_PULL_STR  max length of GUI pulldown string
 *          Z_LEN_GUI_FIELD_STR  max length of field string
 *          Z_LEN_GUI_TITLE_STR   max length of title string
 *
 * * The library is loaded, this routine is called,
 *   then the library is unloaded.
 *
 * * Do not define globals in this routine as when the library is unloaded,
 *   you'll lose them.
 *-------------------------------------------------------------------------------*/
void USERD_get_extra_gui_numbers(int *num_Toggles,
                                 int *num_pulldowns,
                                 int *num_fields)
```

```
/*----------------------------------------------------------------------
 USERD_get_geom_timeset_number
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the timeset number to be used for geometry
 *
 *  It must be in the valid range of timeset numbers
 *       For example,  If USERD_get_number_of_timesets
 *                     returns 2, the valid timeset_number's
 *                     would be 1 and 2.
 *
 *  Returns:
 *  -------
 *  Geom_timeset_number = The timeset number that will be used for geometry.
 *                        For example, if USERD_get_number_of timesets
 *                        returns 2, the valid timeset numbers would be
 *                        1 or 2.
 *
 *  Notes:
 *  * If your model is static, which you indicated by returning a zero
 *    in USERD_get_number_of_timesets, you can return a zero here as well.
 *----------------------------------------------------------------------*/
int
USERD_get_geom_timeset_number( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_gold_part_build_info
 *                                          (version 2.00 and later)
 *                               (Modified at 2.01 as indicated below)
 *                               (Modified at 2.03 as indicated below)
 *----------------------------------------------------------------------
 *   Gets the info needed for part building process
 *
 *  (OUT) part_id                = Array containing the external part
 *                                 ids for each of the model parts.
 *
 *                                 IMPORTANT:
 *                                  External Part ids must be >= 1 because
 *                                  of the way they are used in the GUI
 *
 *             ****************************************
 *              The ids provided here are the numbers by
 *              which the parts will be referred to in the
 *              GUI (if possible). They are basically
 *              labels as far as you are concerned.
 *
 *              Note: The part numbers you pass to routines which receive a
 *              part_number or block_number or which_part as and argument
 *              are the 1-based table index of the parts!
 *
 *              example:  If Numparts_available = 3
 *
 *                        Table index        part_id
 *                        -----------        -------
 *                         1                  13
 *                         2                  57
 *                         3                  125
 *
 *                         ^                  ^
 *                         |                  |
 *                         |                   These are placed in:
 *                         |                     part_id[0] = 13
 *                         |                     part_id[1] = 57
 *                         |                     part_id[2] = 125
 *                         |                   for GUI labeling purposes.
 *                         |
 *                          These implied table indices are the part_number,
 *                          block_number, or which_part numbers that you would
 *                          pass to routines like:
 *
 *                      USERD_get_part_coords(int part_number,...
 *                      USERD_get_part_node_ids(int part_number,...
 *                      USERD_get_part_elements_by_type(int part_number,...
 *                      USERD_get_part_element_ids_by_type(int part_number,...
 *                      USERD_get_block_coords_by_component(int block_number,...
 *                      USERD_get_block_iblanking(int block_number,...
 *                      USERD_get_block_ghost_flags(int block_number,...
 *                      USERD_get_ghosts_in_block_flag(int block_number)
 *                      USERD_get_border_availability(int part_number,...
 *                      USERD_get_border_elements_by_type(int part_number,...
 *                      USERD_get_var_by_component(int which_variable,
 *                                                 int which_part,...
 *                      USERD_get_var_value_at_specific(int which_var,
 *                                                      int which_node_or_elem,
 *                                                      int which_part,...
 *             ****************************************
```

```
*
*                                  (Array will have been allocated
*                                   Numparts_available long)
*
*  (OUT) part_types             = Array containing one of the
*                                 following for each model part:
*
*                                       Z_UNSTRUCTURED or
*                                       Z_STRUCTURED  or
*                                       Z_IBLANKED
*
*                                  (Array will have been allocated
*                                   Numparts_available long)
*
*  (OUT) part_description       = Array containing a description
*                                 for each of the model parts
*
*                                  (Array will have been allocated
*                                   Numparts_available by Z_BUFL
*                                   long)
*
*  (OUT) number_of_nodes        = Number of unstructured nodes in the part
*
*                                  (Array will have been allocated
*                                   Numparts_available long)
*
*  (OUT) number_of_elements     = 2D array containing number of
*                                 each type of element for each
*                                 unstructured model part.
*                                 ------------
*                                 Possible types are:
*
*                                Z_POINT   =  point
*                                Z_BAR02   =  2-noded bar
*                                Z_BAR03   =  3-noded bar
*                                Z_TRI03   =  3-noded triangle
*                                Z_TRI06   =  6-noded triangle
*                                Z_QUA04   =  4-noded quadrilateral
*                                Z_QUA08   =  8-noded quadrilateral
*                                Z_TET04   =  4-noded tetrahedron
*                                Z_TET10   = 10-noded tetrahedron
*                                Z_PYR05   =  5-noded pyramid
*                                Z_PYR13   = 13-noded pyramid
*                                Z_PEN06   =  6-noded pentahedron
*                                Z_PEN15   = 15-noded pentahedron
*                                Z_HEX08   =  8-noded hexahedron
*                                Z_HEX20   = 20-noded hexahedron
*   Starting at API 2.03         Z_NSIDED  = nsided polygon
*   Starting at API 2.03         Z_NFACED  = nfaced polyhedron
*
*   Starting at API 2.01:
*   ====================
*                                Z_G_POINT    ghost node point element
*                                Z_G_BAR02    2 node ghost bar
*                                Z_G_BAR03    3 node ghost bar
*                                Z_G_TRI03    3 node ghost triangle
*                                Z_G_TRI06    6 node ghost triangle
*                                Z_G_QUA04    4 node ghost quad
*                                Z_G_QUA08    8 node ghost quad
*                                Z_G_TET04    4 node ghost tetrahedron
*                                Z_G_TET10   10 node ghost tetrahedron
```

```
*                                Z_G_PYR05    5 node ghost pyramid
*                                Z_G_PYR13   13 node ghost pyramid
*                                Z_G_PEN06    6 node ghost pentahedron
*                                Z_G_PEN15   15 node ghost pentahedron
*                                Z_G_HEX08    8 node ghost hexahedron
*                                Z_G_HEX20   20 node ghost hexahedron
*   Starting at API 2.03         Z_G_NSIDED  ghost nsided polygon
*   Starting at API 2.03         Z_G_NFACED  ghost nfaced polyhedron
*
*                               (Ignored unless Z_UNSTRUCTURED type)
*
*                                  (Array will have been allocated
*                                   Numparts_available by
*                                   Z_MAXTYPE long)
*
*  (OUT) ijk_dimensions          = 2D array containing ijk dimension info
*                                  for structured blocks
*
*                                  For Z_UNSTRUCTURED - is ignored
*
*                                  For Z_STRUCTURED or Z_IBLANKED
*
*       Prior to version 2.03:
*       ----------------------
*                                   (Array will have been allocated
*                                    Numparts_available by 3 long)
*
*                              ijk_dimensions[][0] = I dimension
*                              ijk_dimensions[][1] = J dimension
*                              ijk_dimensions[][2] = K dimension
*
*
*       Starting at version 2.03:
*       -------------------------
*                                    (Array will have been allocated
*                                     Numparts_available by 9 long)
*
*                              There are two ways to do this:
*                              ------------------------------
*                              1. The simple one, without ranges.
*
*                                   This is good for all structured models
*                                   that will a) NOT be used in EnSight's
*                                   Server of Servers. or that will b) be used in
*                                   Server of Servers with EnSight's autodistribute
*                                   option.
*
*                                   Simply provide the ijk dimensions in the
*                                   first three slots and place a -1 in
*                                   the 4th slot.  (The remaining slots will
*                                   be ignored).
*
*                              Thus,
*                              ijk_dimensions[][0] = I dimension of block
*                              ijk_dimensions[][1] = J dimension of block
*                              ijk_dimensions[][2] = K dimension of block
*                              ijk_dimensions[][3] = -1
*
*
*
*
```

```
*
*
*                              example: (Model has one part, a simple 2D block)
*
*                          (J planes)
*                              4 *-------*-------*
*                                |       |       |    ijk_dimension[0][0] = 3
*                                |       |       |    ijk_dimension[0][1] = 4
*                                |       |       |    ijk_dimension[0][2] = 1
*                              3 *-------*-------*
*                                |       |       |    ijk_dimension[0][4] = -1
*                                |       |       |
*                                |       |       |
*                              2 *-------*-------*
*                                |       |       |
*                                |       |       |
*                                |       |       |
*                              1 *-------*-------*
*                                1       2       3  (I planes)
*
*
*
*                              2. Using ranges.
*
*                                  This one can be used anytime, but MUST
*                                  be used if EnSight's Server of Servers
*                                  is to be used without the autodistribute option!
*
*                                  The first 3 slots contain the ijk dimension
*                                  of the complete block (of which this may be
*                                  a portion).  The last 6 slots contain the
*                                  ijk min and max ranges within the complete.
*
*                              Thus,
*                              ijk_dimensions[][0] = I dimension of complete block
*                              ijk_dimensions[][1] = J dimension of complete block
*                              ijk_dimensions[][2] = K dimension of complete block
*
*                              ijk_dimensions[][3] = Imin of portion (1-based)
*                              ijk_dimensions[][4] = Imax of portion (1-based)
*                              ijk_dimensions[][5] = Jmin of portion (1-based)
*                              ijk_dimensions[][6] = Jmax of portion (1-based)
*                              ijk_dimensions[][7] = Kmin of portion (1-based)
*                              ijk_dimensions[][8] = Kmax of portion (1-based)
*
*                              example1: (Model has one part, a simple 2D block,
*                                         and want whole thing)
*
*                          (J planes)
*                              4 *-------*-------*
*                                |       |       |    ijk_dimension[0][0] = 3
*                                |       |       |    ijk_dimension[0][1] = 4
*                                |       |       |    ijk_dimension[0][2] = 1
*                              3 *-------*-------*
*                                |       |       |    ijk_dimension[0][3] = 1
*                                |       |       |    ijk_dimension[0][4] = 3
*                                |       |       |    ijk_dimension[0][5] = 1
*                              2 *-------*-------*    ijk_dimension[0][6] = 4
*                                |       |       |    ijk_dimension[0][7] = 1
*                                |       |       |    ijk_dimension[0][8] = 1
*                                |       |       |
```

```
*                          1 *-------*-------*
*                            1       2       3  (I planes)
* *
*
*                            example2: (Want to have the block represented
*                                       in two portions - 2 parts)
*
*                        (J planes)                  top portion
*                            4 *-------*-------*
*                              |       |       |    ijk_dimension[0][0] = 3
*                              |       |       |    ijk_dimension[0][1] = 4
*                              |       |       |    ijk_dimension[0][2] = 1
*                            3 *-------*-------*
*                              .       .       .    ijk_dimension[0][3] = 1
*                              .       .       .    ijk_dimension[0][4] = 3
*                              .       .       .    ijk_dimension[0][5] = 3
*                            2 .................    ijk_dimension[0][6] = 4
*                              .       .       .    ijk_dimension[0][7] = 1
*                              .       .       .    ijk_dimension[0][8] = 1
*                              .       .       .
*                            1 .................
*                              1       2       3  (I planes)
*
*
*                        (J planes)                  bottom portion
*                            4 .................
*                              .       .       .    ijk_dimension[1][0] = 3
*                              .       .       .    ijk_dimension[2][1] = 4
*                              .       .       .    ijk_dimension[3][2] = 1
*                            3 *-------*-------*
*                              |       |       |    ijk_dimension[1][3] = 1
*                              |       |       |    ijk_dimension[1][4] = 3
*                              |       |       |    ijk_dimension[1][5] = 1
*                            2 *-------*-------*    ijk_dimension[1][6] = 3
*                              |       |       |    ijk_dimension[1][7] = 1
*                              |       |       |    ijk_dimension[1][8] = 1
*                              |       |       |
*                            1 *-------*-------*
*                              1       2       3  (I planes)
*
*
*       And note that if you were partioning this block yourself for
*       EnSight's Server of Servers, you would only have one part,
*       instead of two.  Each SOS server would return its appropriate
*       ranges in the last 6 slots. The first 3 slots would remain constant.
*
*       Just to be clear on the setting of ranges (ijk_dimension[1][3-8],
*         NON-SOS cases:  Ranges can be used to specify partial blocks if desired, or
*                         can be set to the complete block by specifying them, or
*                         by setting ijk_dimension[i][3] to -1
*
*         SOS cases, partitioning in the reader: Ranges must be set properly in this
*                                                routine, so that each server has its
*                                                portion.
*
*         SOS cases, EnSight's autodistribute will be used: Ranges must be set to the
*                                                           complete block for each
*                                                           server. You can use the -1
*                                                           in the ijk_dimension[i][3]
*                                                          slot to do this if desired.
*
```

```
 *
 *  (OUT) iblanking_options      = 2D array containing iblanking
 *                                 options possible for each
 *                                 structured model part.
 *                                 ----------
 *                                 (Ignored unless Z_IBLANKED type)
 *
 *                                 (Array will have been allocated
 *                                  Numparts_available by 6 long)
 *
 *      iblanking_options[][Z_EXT]     = TRUE if external (outside)
 *                       [][Z_INT]     = TRUE if internal (inside)
 *                       [][Z_BND]     = TRUE if boundary
 *                       [][Z_INTBND]  = TRUE if internal boundary
 *                       [][Z_SYM]     = TRUE if symmetry surface
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * If you haven't built a table of pointers to the different parts,
 *    you might want to do so here as you gather the needed info.
 *
 *  * This will be based on Current_time_step
 *-----------------------------------------------------------------------*/
int
USERD_get_gold_part_build_info(int *part_id,
                               int *part_types,
                               char *part_description[Z_BUFL],
                               int *number_of_nodes,
                               int *number_of_elements[Z_MAXTYPE],
                               int *ijk_dimensions[9],
                               int *iblanking_options[6])
```

```
/*----------------------------------------------------------------------
 USERD_get_gold_variable_info
 *                                            (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the variable descriptions, types and filenames
 *
 *  (OUT) var_description      = Variable descriptions
 *
 *                               (Array will have been allocated
 *                                Num_variables by Z_BUFL long)
 *
 *         variable description restrictions:
 *         ----------------------------------
 *         1. Only first 19 characters used in EnSight prior to EnSight 8.2
 *            Starting at EnSight 8.2, 49 characters will be used.
 *         2. Leading and trailing whitespace will be removed by EnSight.
 *         3. Illegal characters will be replaced by underscores.
 *         4. They may not start with a numeric digit.
 *         5. No two variables may have the same description.
 *
 *  (OUT) var_filename         = Variable real filenames
 *
 *                               (Array will have been allocated
 *                                Num_variables by Z_BUFL long)
 *
 *  (OUT) var_type             = Variable type
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *                               types are:  Z_CONSTANT
 *                                           Z_SCALAR
 *                                           Z_VECTOR
 *                                           Z_TENSOR
 *                                           Z_TENSOR9
 *
 *  (OUT) var_classify         = Variable classification
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *                               types are:  Z_PER_NODE
 *                                           Z_PER_ELEM
 *
 *  (OUT) var_complex          = TRUE if complex, FALSE otherwise
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *  (OUT) var_ifilename        = Variable imaginary filenames (if complex)
 *
 *                               (Array will have been allocated
 *                                Num_variables by Z_BUFL long)
 *
 *  (OUT) var_freq             = complex frequency  (if complex)
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *
```

```
 *  (OUT) var_contran          = TRUE if constant changes per time step
 *                               FALSE if constant truly same at all time steps
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *  (OUT) var_timeset          = Timeset the variable will use (1 based).
 *                               (For static models, set it to 1)
 *
 *                               (Array will have been allocated
 *                                Num_variables long)
 *
 *                             For example:  If USERD_get_number_of_timesets
 *                                           returns 2, the valid
 *                                           timeset_number's would be 1 or 2.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The implied variable numbers apply, but be aware that the
 *    arrays are zero based.
 *    So for variable 1, will need to provide   var_description[0]
 *                                              var_filename[0]
 *                                              var_type[0]
 *                                              var_classify[0]
 *                                              var_complex[0]
 *                                              var_ifilename[0]
 *                                              var_freq[0]
 *                                              var_contran[0]
 *                                              var_timeset[0]
 *
 *       for variable 2, will need to provide   var_description[1]
 *                                              var_filename[1]
 *                                              var_type[1]
 *                                              var_classify[1]
 *                                              var_complex[1]
 *                                              var_ifilename[1]
 *                                              var_freq[1]
 *                                              var_contran[1]
 *                                              var_timeset[1]
 *             etc.
 *-------------------------------------------------------------------*/
int
USERD_get_gold_variable_info(char **var_description,
                             char **var_filename,
                             int *var_type,
                             int *var_classify,
                             int *var_complex,
                             char **var_ifilename,
                             float *var_freq,
                             int *var_contran,
                             int *var_timeset)
```

```
/*----------------------------------------------------------------------
 USERD_get_ghosts_in_block_flag
 *                                              (version 2.01 and later)
 *---------------------------------------------------------------------
 *
 *   Gets whether ghost cells present in block or not
 *
 *  (IN) block_number      = The block part number
 *
 *                           (1-based index of part table, namely:
 *
 *                                1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *  returns: TRUE  if any ghost cells in this structured part
 *           FALSE if no ghost cells in this structured part
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Intended for structured parts only, value will be ignored for
 *    unstructured parts
 *---------------------------------------------------------------------*/
int
USERD_get_ghosts_in_block_flag(int block_number)
```

```
/*----------------------------------------------------------------------
  USERD_get_ghosts_in_model_flag
 *                                            (version 2.01 and later)
 *----------------------------------------------------------------------
 *
 *  Answers the question as to whether any ghost cells in the model.
 *
 *  returns:  TRUE        if any ghost cells in the model
 *            FALSE       if no ghost cells in the model
 *
 *  Notes:
 *----------------------------------------------------------------------*/
int
USERD_get_ghosts_in_model_flag( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_matf_escalars_desc
 *                                              (version 2.10 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the list of descriptions of scalar per element variables to be defined
 *  as the specified materials.  This list of element scalar descriptions must be
 *  given in the same order, or sequence as the listed material ids.
 *
 *  (IN)  set_index  = The material set index
 *
 *  (OUT) mesv_desc  = 2D array of per element scalar descriptions
 *                     Array will have beeen allocated by:
 *                     [Num_materials[set_index]] by [Z_BUFL]
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *
 *  Notes:
 *   * See USERD_get_number_of_material_sets header for explanatory example
 *   * Will not be called unless the set_type in USERD_get_matf_set_type
 *     is Z_MISET_VIA_ESCAL_VARS
 *   * Use this in place of the sparse material list interfacing method, which uses
 *     USERD_size_matf_data and USERD_load_matf_data
 *   * This function does not work with material SPECIES, i.e. with the function
 *     calls USERD_get_matsp_info and USERD_get_number_of_species
 *   * The mesv_desc array is NOT the same as the mat_desc array given via the
 *     function USERD_get_matf_var_info
 *----------------------------------------------------------------------*/
int
USERD_get_matf_escalars_desc(int set_index,
                             char **mesv_desc)
```

```
/*----------------------------------------------------------------------
 USERD_get_matf_set_info
 *                                              (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Get the material set ids and names
 *
 *  (OUT) mat_set_ids  = 1D material set ids array.  (A list of unique, non-zero,
 *                                                    positive number ids to be
 *                                                   associated with each material set)
 *
 *                               (Array will have been allocated
 *                                Num_material_sets long)
 *
 *  (OUT) mat_set_name  = 2D material set name array
 *
 *                               (Array will have been allocated
 *                                Num_material_sets by Z_BUFL long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Will not be called if Num_material_sets is zero
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *----------------------------------------------------------------------*/
int
USERD_get_matf_set_info(int *mat_set_ids,
                        char **mat_set_name)
```

```
/*----------------------------------------------------------------------
 USERD_get_matf_set_type
 *                                              (version 2.10 and later)
 *----------------------------------------------------------------------
 *  Gets the material set type
 *
 *  (IN)  set_index  = The material set index
 *
 *  returns: set_type   = Z_MISET_VIA_SPARSE_MIX (default)
 *                        Z_MISET_VIA_ESCAL_VARS
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *  * Z_MISET_VIA_ESCAL_VARS does not work with the material species functions, i.e.
 *    USERD_get_number_of_species and USERD_get_matsp_info
 *  * Use Z_MISET_VIA_SPARSE_MIX to interface materials via sparse material id, mixed
 *    id, and mixed value lists (see USERD_size_matf_data and USRERD_load_matf_data).
 *  * Use Z_MISET_VIA_ESCAL_VARS to interface materials via/as scalar per element
 *    variables (see USERD_get_matf_escalars_desc).
 *----------------------------------------------------------------------*/
int
USERD_get_matf_set_type(int set_index)
```

```
/*----------------------------------------------------------------------
 USERD_get_matf_var_info
 *                                          (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Get the material ids and descriptions for the material set
 *
 *  (IN)  set_index           = the material set index (zero based)
 *
 *  (OUT) mat_ids[set_index]  = 1D materials ids array (internal, non-zero, positive
 *                                                      numeric id)
 *
 *                              (Array will have been allocated
 *                               Num_materials long)
 *
 *  (OUT) mat_desc[set_index] = 2D material descriptions array (description in GUI
 *                                                              material list)
 *
 *                              (Array will have been allocated
 *                               Num_materials by Z_BUFL long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *
 *  * Will not be called if Num_material_sets is zero, or
 *     Num_materials[set_index] is zero
 *----------------------------------------------------------------------*/
int
USERD_get_matf_var_info(int set_index,
                        int *mat_ids,
                        char **mat_desc)
```

```
/*----------------------------------------------------------------------
  USERD_get_matsp_info
 *                                              (version 2.05 and later)
 *----------------------------------------------------------------------
 *
 *   Get the material species ids, descriptions, count per material,
 *   and concatonated lists of species per material for the material set
 *
 *  (IN)  set_index          = Material set index (zero based)
 *
 *  (OUT) sp_ids[set_index]  = 1D material species ids array (non-zero, positive
 *                                                            numeric id)
 *
 *                               (Array will have been allocated
 *                                Num_species long)
 *
 *  (OUT) sp_desc[set_index] = 2D material species descriptions array
 *
 *                               (Array will have been allocated
 *                                Num_species by Z_BUFL long)
 *
 *  (OUT) sp_per_mat_cnt[set_index] = 1D species per material count array
 *
 *                               (Array will have been allocated
 *                                Num_materials long)
 *
 *  (OUT) sp_per_mat_lis[set_index] = 1D concatonated lists of species
 *                                    per material array
 *
 *                               (Array will have been allocated
 *                                Num_materials*Num_species long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *
 *  * Will not be called if Num_material_sets is zero, or
 *     Num_materials[set_index] is zero
 *----------------------------------------------------------------------*/
int
USERD_get_matsp_info(int   set_index,
                     int   *sp_ids,
                     char **sp_desc,
                     int   *sp_per_mat_cnt,
                     int   *sp_per_mat_lis)
```

```
/*----------------------------------------------------------------------
 USERD_get_maxsize_info
 *                                          (version 2.00 and later)
 *                                (Modified at 2.01 as decribed below)
 *----------------------------------------------------------------------
 *
 *   Gets maximum part sizes for efficient memory allocation.
 *
 *   Transient models (especially those that increase in size) can cause
 *   reallocations, at time step changes, to keep chewing up more and
 *   more memory.   The way to avoid this is to know what the maximum
 *   size of such memory will be, and allocate for this maximum initially.
 *
 *   Accordingly, if you choose to provide this information (it is optional),
 *   EnSight will take advantage of it.
 *
 *  (OUT) max_number_of_nodes    = Maximum number of unstructured nodes
 *                                 that will be in the part (over all time).
 *
 *                                  (Array will have been allocated
 *                                   Numparts_available long)
 *
 *  (OUT) max_number_of_elements = 2D array containing maximum number of
 *                                 each type of element for each
 *                                 unstructured model part (over all time).
 *                                 ------------
 *                                 Possible types are:
 *
 *                               Z_POINT   =  point
 *                               Z_BAR02   =  2-noded bar
 *                               Z_BAR03   =  3-noded bar
 *                               Z_TRI03   =  3-noded triangle
 *                               Z_TRI06   =  6-noded triangle
 *                               Z_QUA04   =  4-noded quadrilateral
 *                               Z_QUA08   =  8-noded quadrilateral
 *                               Z_TET04   =  4-noded tetrahedron
 *                               Z_TET10   = 10-noded tetrahedron
 *                               Z_PYR05   =  5-noded pyramid
 *                               Z_PYR13   = 13-noded pyramid
 *                               Z_PEN06   =  6-noded pentahedron
 *                               Z_PEN15   = 15-noded pentahedron
 *                               Z_HEX08   =  8-noded hexahedron
 *                               Z_HEX20   = 20-noded hexahedron
 *
 *   Starting at API 2.01:
 *   ====================
 *                               Z_G_POINT    ghost node point element
 *                               Z_G_BAR02    2 node ghost bar
 *                               Z_G_BAR03    3 node ghost bar
 *                               Z_G_TRI03    3 node ghost triangle
 *                               Z_G_TRI06    6 node ghost triangle
 *                               Z_G_QUA04    4 node ghost quad
 *                               Z_G_QUA08    8 node ghost quad
 *                               Z_G_TET04    4 node ghost tetrahedron
 *                               Z_G_TET10   10 node ghost tetrahedron
 *                               Z_G_PYR05    5 node ghost pyramid
 *                               Z_G_PYR13   13 node ghost pyramid
 *                               Z_G_PEN06    6 node ghost pentahedron
 *                               Z_G_PEN15   15 node ghost pentahedron
 *                               Z_G_HEX08    8 node ghost hexahedron
 *                               Z_G_HEX20   20 node ghost hexahedron
```

```
 *
 *                                (Ignored unless Z_UNSTRUCTURED type)
 *
 *                                   (Array will have been allocated
 *                                    Numparts_available by
 *                                    Z_MAXTYPE long)
 *
 *   (OUT) max_ijk_dimensions = 2D array containing maximum ijk dimensions
 *                              for each structured model part (over_all_time).
 *                                                  ----------
 *                                 (Ignored if Z_UNSTRUCTURED type)
 *
 *                                 (Array will have been allocated
 *                                  Numparts_available by 3 long)
 *
 *                              max_ijk_dimensions[][0] = maximum I dimension
 *                              max_ijk_dimensions[][1] = maximum J dimension
 *                              max_ijk_dimensions[][2] = maximum K dimension
 *
 * Originally:
 *  returns: Z_OK  if supplying maximum data
 *           Z_ERR if not supplying maximum data, or some error occurred
 *                   while trying to obtain it.
 * But as of EnSight 9.0.3(d):
 *  returns: Z_OK  if supplying maximum data (and no largest time step)
 *           largest_time_step + 2 if supplying maximum data and the zero-based
 *                                 time step at which it occurred.
 *                                (The +2 is needed to be able to distinguish this
 *                                 properly from Z_OK or Z_ERR.  It will be subtracted
 *                                 again in EnSight.)
 *           Z_ERR if not supplying maximum data, or some error occurred
 *                   while trying to obtain it.
 *
 *  Notes:
 *  * You need to have first called USERD_get_number_of_model_parts and
 *    USERD_get_gold_part_build_info, so Numparts_available is known and
 *    so EnSight will know what the type is (Z_UNSTRUCTURED, Z_STRUCTURED,
 *    or Z_IBLANKED) of each part.
 *
 *  * This will NOT be based on Current_time_step - it is to be the maximum
 *    values over all time!!
 *
 *  * This information is optional.  If you return Z_ERR, Ensight will still
 *    process things fine, reallocating as needed, etc.  However, for
 *    large transient models you will likely use considerably more memory
 *    and take more processing time for the memory reallocations. So, if it
 *    is possible to provide this information “up front”, it is recommended
 *    to do so.
 *-------------------------------------------------------------------------*/
int
USERD_get_maxsize_info(int *max_number_of_nodes,
                       int *max_number_of_elements[Z_MAXTYPE],
                       int *max_ijk_dimensions[3])
```

```
/*----------------------------------------------------------------------
 USERD_get_model_extents
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the model bounding box extents.  If this routine supplys them
 *   EnSight will not have to spend time doing so.  If this routine
 *   returns Z_ERR, EnSight will have to take the time to touch all the
 *   nodes and gather the extent info.
 *
 *  (OUT) extents[0] = min x
 *                [1] = max x
 *                [2] = min y
 *                [3] = max y
 *                [4] = min z
 *                [5] = max z
 *
 *  returns: Z_ERR if no extents given (EnSight will read all coords and
 *                                      calculate)
 *           Z_OK if extents given
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *----------------------------------------------------------------------*/
int
USERD_get_model_extents( float extents[6] )
```

```
/*----------------------------------------------------------------------
 USERD_get_name_of_reader
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the name of your user defined reader.  The user interface will
 *  ask for this and include it in the available reader list.
 *
 *  (OUT) reader_name          = the name of the reader (data format)
 *                             (max length is Z_MAX_USERD_NAME, which
 *                              is 20)
 *
 *  (OUT) *two_fields          = FALSE if only one data field is required
 *                                      in the data dialog of EnSight.
 *                               TRUE if two data fields required
 *
 *                              -1   if one field (Geom) required
 *                                   and one field (Param) is optional
 *                                   Param field can contain any text
 *                                   for example a file name, modifiers,
 *                                   etc. that can be used to modify the
 *                                   reader's behavior.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Always called.  Provide a name for your custom reader format
 *----------------------------------------------------------------------*/
int
USERD_get_name_of_reader(char reader_name[Z_MAX_USERD_NAME],
                         int *two_fields)
```

```
/*----------------------------------------------------------------------
 USERD_get_nfaced_conn
 *                                          (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the array containing the connectivity of nsided faces of
 *   nfaced elements
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                    loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) nfaced_conn_array  = 1D array of nsided face connectivies of
 *                             nfaced elements
 *
 *                             (int array will have been allocated
 *                              long enough to hold all the nsided
 *                              face connectivities. Which is the sum of
 *                              all the nodes per face values in
 *                              the nfaced_npf_array of
 *                               USERD_get_nfaced_nodes_per_face)
 *
 *   Providing nfaced information to Ensight:
 *
 *   1. In USERD_get_gold_part_build_info, provide the number of nfaced
 *      polyhedral elements in the part.
 *
 *   2. In USERD_get_part_elements_by_type, provide (in the conn_array),
 *      the number of faces per nfaced element. (as if connectivity
 *      length of an nfaced element is one)
 *
 *   3. In this routine, provide the streamed number of nodes per face
 *      for each of the faces of the nfaced elements.
 *
 *
 *   Simple example:          11        10   12
 *                           +---------+-----+
 *   2 nfaced elements:     /|         |\   /|
 *   (1 7-faced            / |         | \ / |
 *    1 5-sided)          /  |         |  +9 |
 *                       /   |         | /|  |
 *                     /7    |       8 /  |  |
 *                    +------------+/ |  |  |
 *                    |      |5    |  |4 |  |6
 *                    |      +-----|--+--|--+
 *                    |    /       |   \ | /
 *                    |   /        |    \|/3
 *                    |  /         |     +
 *                    | /          |    /
 *                    |/1          |2 /
 *                    +------------+/
 *
 *
 *
 *
 *
```

```
*    1. In USERD_get_gold_part_build_info:
*            number_of_elements[Z_NFACED] = 2
*                                            .
*                                           /|\
*                                            |
*    2. In USERD_get_part_elements_by_type:
*        length of conn_array will be:      2 x 1
*        for element_type of Z_NFACED:
*            conn_array[0][0] = 7           (for the 7-faced element)
*            conn_array[1][0] = 5           (for the 5-faced element)
*
*                              ==
*                         Sum  12    <----------+
*                                               |
*    3. In USERD_get_faced_nodes_per_face:      |
*         length of nfaced_npf_array will be:  12
*
*      nfaced_npf_array[0]  = 5  (5-noded top face of 7-faced element)
*      nfaced_npf_array[1]  = 5  (5-noded bot face of 7-faced element)
*      nfaced_npf_array[2]  = 4  (4-noded front face of 7-faced element)
*      nfaced_npf_array[3]  = 4  (4-noded left face of 7-faced element)
*      nfaced_npf_array[4]  = 4  (4-noded back face of 7-faced element)
*      nfaced_npf_array[5]  = 4  (4-noded right front face of 7-faced element)
*      nfaced_npf_array[6]  = 4  (4-noded right back face of 7-faced element)
*
*      nfaced_npf_array[7]  = 3  (3-noded top face of 5-faced element)
*      nfaced_npf_array[8]  = 3  (3-noded bot face of 5-faced element)
*      nfaced_npf_array[9]  = 4  (4-noded back face of 5-faced element)
*      nfaced_npf_array[10] = 4  (4-noded right face of 5-faced element)
*      nfaced_npf_array[11] = 4  (4-noded left front face of 5-faced element)
*
*                            ==
*                       Sum   48   <-----------------+
*                                                    |
*    4. In this function:                            |
*          length of the nfaced_conn_array will be:  48
*
*      nsided_conn_array[0] = 7   (conn of 5-noded top face of 7-faced elem)
*      nsided_conn_array[1] = 8
*      nsided_conn_array[2] = 9
*      nsided_conn_array[3] = 10
*      nsided_conn_array[4] = 11
*
*      nsided_conn_array[5] = 1   (conn of 5-noded bot face of 7-faced elem)
*      nsided_conn_array[6] = 5
*      nsided_conn_array[7] = 4
*      nsided_conn_array[8] = 3
*      nsided_conn_array[9] = 2
*
*      nsided_conn_array[10] = 1  (conn of 4-noded front face of 7-faced elem)
*      nsided_conn_array[11] = 2
*      nsided_conn_array[12] = 8
*      nsided_conn_array[13] = 7
*
*      nsided_conn_array[14] = 5  (conn of 4-noded left face of 7-faced elem)
*      nsided_conn_array[15] = 1
*      nsided_conn_array[16] = 7
*      nsided_conn_array[17] = 11
*
*      nsided_conn_array[18] = 4  (conn of 4-noded back face of 7-faced elem)
*      nsided_conn_array[19] = 5
```

```
 *      nsided_conn_array[20] = 11
 *      nsided_conn_array[21] = 10
 *
 *      nsided_conn_array[22] = 2  (conn of 4-noded right front face of 7-faced)
 *      nsided_conn_array[23] = 3
 *      nsided_conn_array[24] = 9
 *      nsided_conn_array[25] = 8
 *
 *      nsided_conn_array[26] = 3  (conn of 4-noded right back face of 7-faced)
 *      nsided_conn_array[27] = 4
 *      nsided_conn_array[28] = 10
 *      nsided_conn_array[29] = 9
 *
 *      nsided_conn_array[30] = 9  (conn of 3-noded top face of 5-faced elem)
 *      nsided_conn_array[32] = 12
 *      nsided_conn_array[32] = 10
 *
 *      nsided_conn_array[33] = 3  (conn of 3-noded bot face of 5-faced elem)
 *      nsided_conn_array[34] = 4
 *      nsided_conn_array[35] = 6
 *
 *      nsided_conn_array[36] = 6  (conn of 4-noded back face of 5-faced elem)
 *      nsided_conn_array[37] = 4
 *      nsided_conn_array[38] = 10
 *      nsided_conn_array[39] = 12
 *
 *      nsided_conn_array[40] = 3  (conn of 4-noded right face of 5-faced elem)
 *      nsided_conn_array[41] = 6
 *      nsided_conn_array[42] = 12
 *      nsided_conn_array[43] = 9
 *
 *      nsided_conn_array[44] = 4  (conn of 4-noded left front face of 5-faced)
 *      nsided_conn_array[45] = 3
 *      nsided_conn_array[46] = 9
 *      nsided_conn_array[47] = 10
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Will not be called unless there are some nfaced elements in the
 *    the part
 *----------------------------------------------------------------------*/
int
USERD_get_nfaced_conn(int part_number,
                      int *nfaced_conn_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_nfaced_conn_in_buffers
 *                                <optional>  (version 2.08 and later)
 *----------------------------------------------------------------------
 *
 *   Gets three arrays containing the number of faces per element,
 *   number of nodes per face, and connectivity per face of nfaced
 *   elements in buffers
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute capability
 *   in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines) in
 *   your reader (in a proper manner), you should be able to not only get the
 *   execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_nfaced_conn routine. And thus its implementation should
 *   not be too difficult to add to any existing reader that has already
 *   implemented the USERD_get_nfaced_conn routine.
 *
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  first            = TRUE if first invocation of a buffered set.
 *                           Will be FALSE for all subsequent invocations
 *                           of the set.  This is so you can open files,
 *                           get to the correct starting spot,
 *                           initialize, etc.
 *
 *  (IN) e_beg             = Zero based, first element number
 *                           of the buffered set
 *
 *  (IN) e_end             = Zero based, last element number
 *                           of the buffered set
 *
 *                     Thus, for first five elements of a type:
 *                       e_beg = 0
 *                       e_end = 4
 *                       total_number = (e_end - e_beg) + 1 = (4 - 0) + 1 = 5
 *
 *
```

```
*                     for second five elements of a type, would be:
*                       e_beg = 5
*                       e_end = 9
*                       total_number = (e_end - e_beg) + 1 = (9 - 5) + 1 = 5
*
*                     for all elements of the type of a part, would be:
*                       n_beg = 0
*                       n_end = num_elements_of_type - 1
*
*  (IN) buffer_size         = The size of the num_nodes_per_elem_array buffer.
*                             Namely: num_nodes_per_elem_array[buffer_size]
*
*  (OUT) nfaced_fpe_array   = 1D buffer array of the number of faces per nfaced
*                             element.
*
*                                (int array will have been allocated
*                                 buffer_size long)
*
*  (OUT) nfaced_npf_array   = 1D buffer array of the number of nodes per face
*                             for nfaced elements.
*
*                                (int array will have been allocated long
*                                 enough to hold a buffer's size of values)
*
*  (OUT) nfaced_conn_array  = 1D array of nsided face connectivies of
*                             nfaced elements
*
*                              (int array will have been allocated
*                               long enough to hold a buffer's worth of values)
*
*   Providing nfaced information to Ensight:
*
*   NOTE: for other nfaced operations you need these first two, but we
*         don't actually use them in this routine.
*
*   1. In USERD_get_gold_part_build_info, provide the number of nfaced
*      polyhedral elements in the part.
*
*   2. In USERD_get_part_elements_by_type, provide (in the conn_array),
*      the number of faces per nfaced element. (as if connectivity
*      length of an nfaced element is one)
*
*   We do use the following:
*   3. In this routine, provide the corresponding number of faces per nfaced
*      element, streamed number of nodes per face, and streamed face
*      connectivities for each of the faces of the nfaced elements in the
*      bufferred portion.
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
```

```
*   Simple example:          11        10   12
*                           +---------+-----+
*   2 nfaced elements:     /|         |\   /|
*   (1 7-faced            / |         | \ / |
*    1 5-sided)          /  |         |  +9 |
*                       /   |         | /|  |
*                      /7   |       8 /  |  |
*                     +-----------+/ |  |  |
*                     |     |5    |  |4 |  |6
*                     |     +-----|--+--|--+
*                     |    /      |   \ | /
*                     |   /       |    \|/3
*                     |  /        |     +
*                     | /         |    /
*                     |/1         |2 /
*                     +-----------+/
*
*    Note, don't really use these first two here (See USERD_get_nfaced_conn)
*
*    1. In USERD_get_gold_part_build_info:
*           number_of_elements[Z_NFACED] = 2
*                                           .
*                                          /|\
*                                           |
*    2. In USERD_get_part_elements_by_type:
*        length of conn_array will be:      2 x 1
*        for element_type of Z_NFACED:
*            conn_array[0][0] = 7           (for the 7-faced element)
*            conn_array[1][0] = 5           (for the 5-faced element)
*                              ==
*                         Sum  12
*
*
*    But for our simple example, lets assume that that our buffer is just 1
*      so that we have multiple invocations.              ================
*    3. In this routine:
*
*       first invocation:
*         first = TRUE
*         e_beg = 0
*         e_end = 1
*         buffer_size = 1
*         nfaced_fpe_array[1]            load it: nfaced_fpe_array[0] = 7
*
*         nfaced_npf_array[at least 7]   load it: nfaced_npf_array[0] = 5
*                                                 nfaced_npf_array[1] = 5
*                                                 nfaced_npf_array[2] = 4
*                                                 nfaced_npf_array[3] = 4
*                                                 nfaced_npf_array[4] = 4
*                                                 nfaced_npf_array[5] = 4
*                                                 nfaced_npf_array[6] = 4
*
*         nsided_conn_array[at least 30] load it: nsided_conn_array[0] = 7
*                                                 nsided_conn_array[1] = 8
*                                                 nsided_conn_array[2] = 9
*                                                 nsided_conn_array[3] = 10
*                                                 nsided_conn_array[4] = 11
*
*                                                 nsided_conn_array[5] = 1
*                                                 nsided_conn_array[6] = 5
*                                                 nsided_conn_array[7] = 4
```

```
*                                                nsided_conn_array[8] = 3
*                                                nsided_conn_array[9] = 2
*
*                                                nsided_conn_array[10] = 1
*                                                nsided_conn_array[11] = 2
*                                                nsided_conn_array[12] = 8
*                                                nsided_conn_array[13] = 7
*
*                                                nsided_conn_array[14] = 5
*                                                nsided_conn_array[15] = 1
*                                                nsided_conn_array[16] = 7
*                                                nsided_conn_array[17] = 11
*
*                                                nsided_conn_array[18] = 4
*                                                nsided_conn_array[19] = 5
*                                                nsided_conn_array[20] = 11
*                                                nsided_conn_array[21] = 10
*
*                                                nsided_conn_array[22] = 2
*                                                nsided_conn_array[23] = 3
*                                                nsided_conn_array[24] = 9
*                                                nsided_conn_array[25] = 8
*
*                                                nsided_conn_array[26] = 3
*                                                nsided_conn_array[27] = 4
*                                                nsided_conn_array[28] = 10
*                                                nsided_conn_array[29] = 9
*         *num_returned = 1;
*         return(0)
*
*       second invocation:
*         first = FALSE
*         e_beg = 0
*         e_end = 1
*         buffer_size = 1
*         nfaced_fpe_array[1]            load it: nfaced_fpe_array[0] = 5
*
*         nfaced_npf_array[at least 7]   load it: nfaced_npf_array[0] = 3
*                                                 nfaced_npf_array[1] = 3
*                                                 nfaced_npf_array[2] = 4
*                                                 nfaced_npf_array[3] = 4
*                                                 nfaced_npf_array[4] = 4
*
*         nsided_conn_array[at least 18] load it: nsided_conn_array[0] = 9
*                                                 nsided_conn_array[1] = 12
*                                                 nsided_conn_array[2] = 10
*
*                                                 nsided_conn_array[3] = 3
*                                                 nsided_conn_array[4] = 4
*                                                 nsided_conn_array[5] = 6
*
*                                                 nsided_conn_array[6] = 6
*                                                 nsided_conn_array[7] = 4
*                                                 nsided_conn_array[8] = 10
*                                                 nsided_conn_array[9] = 12
*
*                                                 nsided_conn_array[10] = 3
*                                                 nsided_conn_array[11] = 6
*                                                 nsided_conn_array[12] = 12
*                                                 nsided_conn_array[13] = 9
*
```

```
 *                                                  nsided_conn_array[14] = 4
 *                                                  nsided_conn_array[15] = 3
 *                                                  nsided_conn_array[16] = 9
 *                                                  nsided_conn_array[17] = 10
 *         *num_returned = 1;
 *         return(1)
 *
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Will not be called unless there are some nfaced elements in the
 *    the part
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_nfaced_conn_in_buffers(int part_number,
                                 int *nfaced_fpe_array,
                                 int *nfaced_npf_array,
                                 int *nfaced_conn_array,
                                 int first,
                                 int e_beg,
                                 int e_end,
                                 int buffer_size,
                                 int *num_returned)
```

```
/*----------------------------------------------------------------------
 USERD_get_nfaced_nodes_per_face
 *                                          (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the array containing the number of nodes per face for each face
 *   of the nfaced elements.
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) nfaced_npf_array  = 1D array of nodes per face for all
 *                            faces of nfaced elements
 *
 *                             (int array will have been allocated
 *                              long enough to hold all the nodes per
 *                              face values. Which is the sum of
 *                              all the number of faces per element values in
 *                              the conn_array of
 *                              USERD_get_part_elements_by_type)
 *
 *  Providing nfaced information to Ensight:
 *
 *   1. In USERD_get_gold_part_build_info, provide the number of nfaced
 *      polyhedral elements in the part.
 *
 *   2. In USERD_get_part_elements_by_type, provide (in the conn_array),
 *      the number of faces per nfaced element. (as if connectivity
 *      length of an nfaced element is one)
 *
 *   3. In this routine, provide the streamed number of nodes per face
 *      for each of the faces of the nfaced elements.
 *
 *
 *   Simple example:          11        10   12
 *                           +--------+-----+
 *   2 nfaced elements:     /|        |\   /|
 *   (1 7-faced            / |        | \ / |
 *    1 5-sided)          /  |        |  +9 |
 *                       /   |        | /|  |
 *                     /7    |      8 /  |  |
 *                    +------------+/ |  |  |
 *                    |      |5    |  |4 |  |6
 *                    |      +-----|--+--|--+
 *                    |    /       |   \ | /
 *                    |   /        |    \|/3
 *                    |  /         |     +
 *                    | /          |    /
 *                    |/1          |2 /
 *                    +------------+/
 *
 *
 *
 *
 *
```

```
*    1. In USERD_get_gold_part_build_info:
*            number_of_elements[Z_NFACED] = 2
*                                            .
*                                           /|\
*                                            |
*    2. In USERD_get_part_elements_by_type:
*        length of conn_array will be:      2 x 1
*        for element_type of Z_NFACED:
*            conn_array[0][0] = 7           (for the 7-faced element)
*            conn_array[1][0] = 5           (for the 5-faced element)
*
*                               ==
*                         Sum  12    <----------+
*                                               |
*    3. In this routine:                        |
*         length of nfaced_npf_array will be:  12
*
*      nfaced_npf_array[0]  = 5  (5-noded top face of 7-faced element)
*      nfaced_npf_array[1]  = 5  (5-noded bot face of 7-faced element)
*      nfaced_npf_array[2]  = 4  (4-noded front face of 7-faced element)
*      nfaced_npf_array[3]  = 4  (4-noded left face of 7-faced element)
*      nfaced_npf_array[4]  = 4  (4-noded back face of 7-faced element)
*      nfaced_npf_array[5]  = 4  (4-noded right front face of 7-faced element)
*      nfaced_npf_array[6]  = 4  (4-noded right back face of 7-faced element)
*
*      nfaced_npf_array[7]  = 3  (3-noded top face of 5-faced element)
*      nfaced_npf_array[8]  = 3  (3-noded bot face of 5-faced element)
*      nfaced_npf_array[9]  = 4  (4-noded back face of 5-faced element)
*      nfaced_npf_array[10] = 4  (4-noded right face of 5-faced element)
*      nfaced_npf_array[11] = 4  (4-noded left front face of 5-faced element)
*
*                             ==
*                       Sum   48   <-----------------+
*                                                    |
*    4. In USERD_get_nfaced_conn:                    |
*          length of the nfaced_conn_array will be:  48
*
*      nsided_conn_array[0] = 7   (conn of 5-noded top face of 7-faced elem)
*      nsided_conn_array[1] = 8
*      nsided_conn_array[2] = 9
*      nsided_conn_array[3] = 10
*      nsided_conn_array[4] = 11
*
*      nsided_conn_array[5] = 1   (conn of 5-noded bot face of 7-faced elem)
*      nsided_conn_array[6] = 5
*      nsided_conn_array[7] = 4
*      nsided_conn_array[8] = 3
*      nsided_conn_array[9] = 2
*
*      nsided_conn_array[10] = 1  (conn of 4-noded front face of 7-faced elem)
*      nsided_conn_array[11] = 2
*      nsided_conn_array[12] = 8
*      nsided_conn_array[13] = 7
*
*      nsided_conn_array[14] = 5  (conn of 4-noded left face of 7-faced elem)
*      nsided_conn_array[15] = 1
*      nsided_conn_array[16] = 7
*      nsided_conn_array[17] = 11
*
*      nsided_conn_array[18] = 4  (conn of 4-noded back face of 7-faced elem)
*      nsided_conn_array[19] = 5
```

```
 *      nsided_conn_array[20] = 11
 *      nsided_conn_array[21] = 10
 *
 *      nsided_conn_array[22] = 2  (conn of 4-noded right front face of 7-faced)
 *      nsided_conn_array[23] = 3
 *      nsided_conn_array[24] = 9
 *      nsided_conn_array[25] = 8
 *
 *      nsided_conn_array[26] = 3  (conn of 4-noded right back face of 7-faced)
 *      nsided_conn_array[27] = 4
 *      nsided_conn_array[28] = 10
 *      nsided_conn_array[29] = 9
 *
 *      nsided_conn_array[30] = 9  (conn of 3-noded top face of 5-faced elem)
 *      nsided_conn_array[32] = 12
 *      nsided_conn_array[32] = 10
 *
 *      nsided_conn_array[33] = 3  (conn of 3-noded bot face of 5-faced elem)
 *      nsided_conn_array[34] = 4
 *      nsided_conn_array[35] = 6
 *
 *      nsided_conn_array[36] = 6  (conn of 4-noded back face of 5-faced elem)
 *      nsided_conn_array[37] = 4
 *      nsided_conn_array[38] = 10
 *      nsided_conn_array[39] = 12
 *
 *      nsided_conn_array[40] = 3  (conn of 4-noded right face of 5-faced elem)
 *      nsided_conn_array[41] = 6
 *      nsided_conn_array[42] = 12
 *      nsided_conn_array[43] = 9
 *
 *      nsided_conn_array[44] = 4  (conn of 4-noded left front face of 5-faced)
 *      nsided_conn_array[45] = 3
 *      nsided_conn_array[46] = 9
 *      nsided_conn_array[47] = 10
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Will not be called unless there are some nfaced elements in the
 *    the part
 *----------------------------------------------------------------------*/
int
USERD_get_nfaced_nodes_per_face(int part_number,
                                int *nfaced_npf_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_node_label_status
 *                                          (version 2.00 and later)
 *                                (Modified at 2.01 as described below)
 *----------------------------------------------------------------------
 *
 *  Answers the question as to whether node labels will be provided.
 *
 *  returns:  TRUE        if node labels will be provided
 *            FALSE       if node labels will NOT be provided
 *
 *  Notes:
 *  * These are needed in order to do any node querying, or node
 *    labeling on-screen                                    .
 *
 * * Will call USERD_get_part_node_ids for each part if this routine
 *   returns TRUE.
 *
 *  * Prior to API 2.01:
 *    =================
 *      For unstructured parts, you can read them from your file if
 *      available, or can assign them, etc. They need to be unique
 *      per part, and are often unique per model. They must also be
 *      positive numbers greater than zero.
 *
 *        USERD_get_part_node_ids is used to obtain the ids, if the
 *        status returned here is TRUE.
 *
 *        (Unlike API 1.0, where the connectivity of elements had to be
 *         according to the node ids - API 2.0's element connectivities
 *         are not affected either way by the status here.)
 *
 *      For structured parts, EnSight will assign ids if you return a
 *        status of TRUE here.  You cannot assign them yourself!!
 *
 *  * Starting at API 2.01:
 *    ====================
 *      For both unstructured and structured parts, you can read them
 *      from your file if available, or can assign them, etc. They need
 *      to be unique per part, and are often unique per model. They must
 *      also be positive numbers greater than zero.
 *
 *        USERD_get_part_node_ids is used to obtain the ids, if the
 *        status returned here is TRUE.
 *----------------------------------------------------------------------*/
int
USERD_get_node_label_status( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_nsided_conn
 *                                              (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the array containing the connectivity of nsided elements
 *
 *  (IN)  part_number      = The part number
 *
 *                             (1-based index of part table, namely:
 *
 *                                1 ... Numparts_available.
 *
 *                              It is NOT the part_id that
 *                              is loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) nsided_conn_array  = 1D array of nsided connectivies
 *
 *                               (int array will have been allocated
 *                                long enough to hold all the nsided
 *                                connectivities. Which is the sum of
 *                                all the nodes per element values in
 *                                the conn_array of
 *                                USERD_get_part_elements_by_type)
 *
 *   Providing nsided information to Ensight:
 *
 *   1. In USERD_get_gold_part_build_info, provide the number of nsided
 *      elements in the part.
 *
 *   2. In USERD_get_part_elements_by_type, provide (in the conn_array),
 *      the number of nodes per nsided element. (as if connectivity
 *      length of an nsided element is one)
 *
 *   3. In this routine, provide the streamed connectivities for each of the
 *      nsided elements.
 *
 *
 *   Simple example:          5        6
 *                           +--------+
 *   3 nsided elements:     /|         \
 *   (1 4-sided            / |          \
 *    1 3-sided           /  |           \
 *    1 7-sided)         /   |            \ 7
 *                      /3   |4            +
 *                     +-----+             |
 *                     |     |             |
 *                     |     |             |8
 *                     |     |             +
 *                     |     |            /
 *                     |     |           /
 *                     |     |          /
 *                     |1    |2        /9
 *                     +-----+--------+
 *
 *
 *
 *
 *
 *
 *
```

```
 *    1. In USERD_get_gold_part_build_info:
 *            number_of_elements[Z_NSIDED] = 3
 *                                            .
 *                                           /|\
 *                                            |
 *    2. In USERD_get_part_elements_by_type:
 *        length of conn_array will be:      3 x 1
 *
 *        for element_type of Z_NSIDED:
 *            conn_array[0][0] = 4           (for the 4-sided element)
 *            conn_array[1][0] = 3           (for the 3-sided element)
 *            conn_array[2][0] = 7           (for the 7-sided element)
 *
 *                         Sum  ===
 *                               14    <---------+
 *                                               |
 *    3. In this routine:                        |
 *         length of nsided_conn_array will be:  14
 *
 *            nsided_conn_array[0]  = 1      (connectivity of 4-sided element)
 *            nsided_conn_array[1]  = 2
 *            nsided_conn_array[2]  = 4
 *            nsided_conn_array[3]  = 3
 *
 *            nsided_conn_array[4]  = 3      (connectivity of 3-sided element)
 *            nsided_conn_array[5]  = 4
 *            nsided_conn_array[6]  = 5
 *
 *            nsided_conn_array[7]  = 2      (connectivity of 7-sided element)
 *            nsided_conn_array[8]  = 9
 *            nsided_conn_array[9]  = 8
 *            nsided_conn_array[10] = 7
 *            nsided_conn_array[11] = 6
 *            nsided_conn_array[12] = 5
 *            nsided_conn_array[13] = 4
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Will not be called unless there are some nsided elements in the
 *    the part
 *-------------------------------------------------------------------------*/
int
USERD_get_nsided_conn(int part_number,
                      int *nsided_conn_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_nsided_conn_in_buffers
 *                                  <optional> (version 2.08 or later)
 *----------------------------------------------------------------------
 *   Gets the two arrays containing the connectivity information
 *   of nsided elements in buffers
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines) in
 *   your reader (in a proper manner), you should be able to not only get the
 *   execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_nsided_conn routine. And thus its implementation should
 *   not be too difficult to add to any existing reader that has already
 *   implemented the USERD_get_nsided_conn routine.
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  first            = TRUE if first invocation of a buffered set.
 *                           Will be FALSE for all subsequent invocations
 *                           of the set.  This is so you can open files,
 *                           get to the correct starting spot,
 *                           initialize, etc.
 *
 *  (IN) e_beg             = Zero based, first element number
 *                           of the buffered set
 *
 *  (IN) e_end             = Zero based, last element number
 *                           of the buffered set
 *
 *                     Thus, for first five elements of a type:
 *                       e_beg = 0
 *                       e_end = 4
 *                       total_number = (e_end - e_beg) + 1 = (4 - 0) + 1 = 5
 *
 *                     for second five elements of a type, would be:
 *                       e_beg = 5
 *                       e_end = 9
 *                       total_number = (e_end - e_beg) + 1 = (9 - 5) + 1 = 5
```

```
*
*                     for all elements of the type of a part, would be:
*                       n_beg = 0
*                       n_end = num_elements_of_type - 1
*
*   (IN) buffer_size         = The size of the num_nodes_per_elem_array buffer.
*                              Namely: num_nodes_per_elem_array[buffer_size]
*
*   (OUT) num_nodes_per_elem_array = 1D buffer array of the number of nodes
*                                    per nsided element.
*
*   (OUT) nsided_conn_array  = 1D buffer array of nsided connectivies
*
*                              (int array will have been allocated
*                               long enough to hold all the nsided
*                               connectivities in the buffered chunk)
*
*   (OUT) *num_returned      = The number of elements whose connectivities
*                              are returned in the buffer. This will
*                              normally be equal to buffer_size except for
*                              that last buffer - which could be less than
*                              a full buffer.
*
*    Providing nsided information to Ensight:
*
*    NOTE: for other nsided operations you need these first two, but we
*          don't actually use them in this routine.
*
*    1. In USERD_get_gold_part_build_info, provide the number of nsided
*       elements in the part.
*
*    2. In USERD_get_part_elements_by_type, provide (in the conn_array),
*       the number of nodes per nsided element. (as if connectivity
*       length of an nsided element is one)
*
*    We do use the following:
*    3. In this routine, provide the corresponding num_nodes_per_element and
*       streamed connectivities for each of the nsided elements in this
*       buffered portion.
*
*
*    Simple example:          5        6
*                            +--------+
*    3 nsided elements:     /|         \
*    (1 4-sided            / |          \
*     1 3-sided           /  |           \
*     1 7-sided)         /   |            \ 7
*                       /3   |4            +
*                      +-----+             |
*                      |     |             |
*                      |     |             |8
*                      |     |             +
*                      |     |            /
*                      |     |           /
*                      |     |          /
*                      |1    |2        /9
*                      +-----+--------+
*
*
*
*
```

```
*    NOTE, don't really use these first two here  (See USERD_get_nsided_conn)
*
*    1. In USERD_get_gold_part_build_info:
*            number_of_elements[Z_NSIDED] = 3
*                                            .
*                                           /|\
*                                            |
*    2. In USERD_get_part_elements_by_type:
*        length of conn_array will be:      3 x 1
*
*        for element_type of Z_NSIDED:
*            conn_array[0][0] = 4           (for the 4-sided element)
*            conn_array[1][0] = 3           (for the 3-sided element)
*            conn_array[2][0] = 7           (for the 7-sided element)
*
*                          Sum  ===
*                                14
*
*    But for our example, lets assume that that our buffer is just 2
*                                                 ================
*    3. In this routine:
*
*       first invocation:
*         first = TRUE
*         e_beg = 0
*         e_end = 2
*         buffer_size = 2
*         num_nodes_per_elem_array[2]    load it:
*                                            num_nodes_per_elem_array[0] = 4
*                                            num_nodes_per_elem_array[1] = 3
*
*         nsided_conn_array[at least 7]  load it:   nsided_conn_array[0] = 1
*                                                   nsided_conn_array[1] = 2
*                                                   nsided_conn_array[2] = 4
*                                                   nsided_conn_array[3] = 3
*
*                                                   nsided_conn_array[4] = 3
*                                                   nsided_conn_array[5] = 4
*                                                   nsided_conn_array[6] = 5
*         *num_returned = 2
*         return(0)                       return this (indicates more to do)
*
*       second invocation:
*         first = FALSE
*         e_beg = 0
*         e_end = 2
*         buffer_size = 2
*         num_nodes_per_elem_array[2]    load it:
*                                            num_nodes_per_elem_array[0] = 7
*
*         nsided_conn_array[at least 7]  load it:   nsided_conn_array[0] = 2
*                                                   nsided_conn_array[1] = 9
*                                                   nsided_conn_array[2] = 8
*                                                   nsided_conn_array[3] = 7
*                                                   nsided_conn_array[4] = 6
*                                                   nsided_conn_array[5] = 5
*                                                   nsided_conn_array[6] = 4
*         *num_returned = 1
*         return(1)                       return this (indicates no more to do)
*
*
```

```
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Will not be called unless there are some nsided elements in the
 *    the part
 *-----------------------------------------------------------------------*/
int
USERD_get_nsided_conn_in_buffers(int part_number,
                                 int *num_nodes_per_elem_array,
                                 int *nsided_conn_array,
                                 int first,
                                 int e_beg,
                                 int e_end,
                                 int buffer_size,
                                 int *num_returned)
```

```
/*----------------------------------------------------------------------
 USERD_get_num_of_time_steps
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the number of time steps of data available for desired timeset.
 *
 *  (IN)  timeset_number       = the timeset number  (1 based)
 *
 *                               For example: If USERD_get_number_of_timesets
 *                                            returns 2, the valid
 *                                            timeset_number's would be 1 and 2.
 *
 *  returns: number of time steps (>0 if okay, <=0 if problems).
 *
 *  Notes:
 *  * This should be >= 1       1 indicates a static timeset
 *                             >1 indicates a transient problem
 *
 *----------------------------------------------------------------------*/
int
USERD_get_num_of_time_steps( int timeset_number )
```

```
/*-----------------------------------------------------------------------
 USERD_get_num_xy_queries
 *                               <optional>   (version 2.08 and later)
 *-----------------------------------------------------------------------
 *
 *   Get the total number of xy queries in the dataset.
 *
 *  returns: the total number of xy queries in the dataset
 *
 *  Notes:
 *  * You can be as complete as you want about this.  If you don't
 *    care about xy queries, return a value of 0
 *    If you only want certain xy queries, you can just include them. But,
 *    you will need to supply the info and data USERD_get_xy_query_info
 *    and USERD_get_xy_query_data for each xy query you include here.
 *-----------------------------------------------------------------------*/
int
USERD_get_num_xy_queries( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_files_in_dataset
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the total number of files in the dataset.  Used for the
 *   dataset query option.
 *
 *  returns: the total number of files in the dataset
 *
 *  Notes:
 *  * You can be as complete as you want about this.  If you don't
 *    care about the dataset query option, return a value of 0
 *    If you only want certain files, you can just include them. But,
 *    you will need to supply the info in USERD_get_dataset_query_file_info
 *    for each file you include here.
 *
 *  * Num_dataset_files would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_files_in_dataset( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_material_sets
 *                                                  (version 2.03 and later)
 *----------------------------------------------------------------------
 *   Get the number of material sets in the model
 *
 *  returns: Num_material_sets = number of material sets
 *                              (Zero would indicate that you have no materials
 *                               to deal with in the model)
 *              or
 *
 *          -1 if an error condition
 *  Notes:
 *
 *  * You may want to keep this as a global for use in other routines.
 *
 *   ################################################################
 *   NOTE:  Since EnSight 7.6, only one material set is supported within EnSight
 *          Thus the only valid returns here are:
 *               0 (no materials)
 *               1 (for the one material set allowed)
 *          or  -1 (if an error)
 *
 *          If the casefile has more than this, this reader will read them,
 *          but EnSight will issue an error message and choke on them!
 *   ################################################################
 *
 *  ================================================================
 *  A very simple explanatory example, to use as a reference for the materials routines:
 *
 *  Given a 2D mesh composed of 9 quad (Z_QUA04) elements, with two materials.
 *  Most of the model is material 1, but the top left corner is material 9 -
 *  basically as shown:
 *
 *        *--------*--------*--------*
 *        |        |   /    |        |
 *        |     Mat 9 /     |        |
 *        |        | /      |        |
 *        |        |/       |        |
 *        |  e7    /   e8   |   e9   |
 *        |       /|        |        |
 *        |      / |        |        |
 *        |     /  |        |        |
 *        *----/---*--------*--------*
 *        |   /    |        |        |
 *        |  /     |        |        |
 *        | /      |      Mat 1      |
 *        |/       |        |        |
 *        |   e4   |   e5   |   e6   |
 *        |        |        |        |
 *        |        |        |        |
 *        |        |        |        |
 *        *--------*--------*--------*
 *        |        |        |        |
 *        |        |        |        |
 *        |        |        |        |
 *        |   e1   |   e2   |   e3   |
 *        |        |        |        |
 *        |        |        |        |
 *        |        |        |        |
 *        *--------*--------*--------*
```

```
*
*  Thus, in this routine, set:
*    Num_material_sets = 1
*
*  In USERD_get_matf_set_info, set:
*    mat_set_ids[0]    = 1
*    mat_set_name[0]   = "Material Set 1"  (or whatever name desired)
*
*  In USERD_get_number_of_materials, input would be set_index = 0, and
*  would need to set:
*    Num_materials[0] = 2
*
*  For simplicity, the ids and descriptions that would be returned in
*  USERD_get_matf_var_info could be:
*    mat_ids[0] = 1
*    mat_ids[1] = 9
*    mat_desc[0] = "mat 1"   (or whatever desired)
*    mat_desc[2] = "mat 9"
*
*  Note1: In USERD_get_matf_set_type, the input would be set_index = 0, and if
*         set_type = Z_MISET_VIA_SPARSE_MIX, the following 3 lists (material ids,
*         mixed-material ids, and mixed-material values) would be needed and
*         communicated by the following two USERD calls, i.e. USERD_size_matf_data
*         and USERD_load_matf_data.
*
*  The per element material ids list would need to be:
*
*     material ids:
*     -------------
*     ids_list[0] = 1  (material id 1, for elem e1)
*     ids_list[1] = 1  (     "                  e2)
*     ids_list[2] = 1  (     "                  e3)
*     ids_list[3] = -1 (neg. of index into mixed-material id list, for elem e4)
*     ids_list[4] = 1  (material id 1, for elem e5)
*     ids_list[5] = 1  (     "                  e6)
*     ids_list[6] = -5 (neg. of index into mixed-material id list, for elem e7)
*     ids_list[7] = -9 (     "                                          e8)
*     ids_list[8] = 1  (material id 1, for elem e9)
*
*  Finally we need the mixed material ids list and the mixed materials
*  values list, which would need to be:
*
*       mixed-material ids:
*       -------------------
*    ==> 1  ids_list[0]  =  2  (the -1 in the material variable points here,
*                               2 indicates that two materials are present)
*       2  ids_list[1]  =  1  (1st material is 1)
*       3  ids_list[2]  =  9  (2nd material is 9)
*       4  ids_list[3]  = -1  (negative of index into mixed-material val_list)
*    ==> 5  ids_list[4]  =  2  (the -5 in the material variable points here,
*                               2 indicates that two materials are present)
*       6  ids_list[5]  =  1  (1st material is 1)
*       7  ids_list[6]  =  9  (2nd material is 9)
*       8  ids_list[7]  = -3  (negative of index into mixed-material val_list)
*    ==> 9  ids_list[8]  =  2     etc.
*       10 ids_list[9]  =  1
*       11 ids_list[10] =  9
*       12 ids_list[11] = -5
*
*       mixed-material values:
*       ----------------------
```

```
*   ==> 1 val_list[0] = 0.875 (the -1 in the  mixed-material ids_list points
*                             here, and this is the value for material 1)
*       2 val_list[1] = 0.125 (the value for material 9)
*   ==> 3 val_list[2] = 0.125 (the -3 in the mixed-materials ids_list points
*                             here)
*       4 val_list[3] = 0.875
*   ==> 5 val_list[4] = 0.875 (the -5 in the mixed-materials ids_list points
*                             here)
*       6 val_list[5] = 0.125
*
*  So, USERD_size_matf_data would need to return
*       matf_size = 8, when called with set_id    = 1
*                                       part_id   = 1
*                                       wtyp      = Z_QUA04
*                                       mat_type  = Z_MAT_INDEX
*
*       matf_size = 12, when called with set_id   = 1
*                                        part_id  = 1
*                                        mat_type = Z_MIX_INDEX
*
*                 = 6, when called with set_id   = 1
*                                       part_id  = 1
*                                       mat_type = Z_MIX_VALUE
*
*  And, USERD_load_matf_data would need to return:
*    the int array ids_list as shown above when called with:
*       set_id   = 1
*       part_id  = 1
*       wtyp     = Z_QUA04
*       mat_type = Z_MAT_INDEX (indicating id list).
*
*    the int array ids_list as shown above when called with:
*       set_id   = 1
*       part_id  = 1
*       mat_type = Z_MIX_INDEX (indicating id list).
*
*    the float array val_list as shown above when called with:
*       set_id   = 1
*       part_id  = 1
*       mat_type = Z_MIX_VALUE (indicating val list).
*
* Note2: In USERD_get_matf_set_type, the input would be set_index = 0, and if
*        set_type = Z_MISET_VIA_ESCAL_VARS, then only USERD_get_matf_escalars_desc
*        would be called.
*
*        In USERD_get_matf_escalars_desc, the input would be set_index = 0, and the
*        descriptions of the per element scalar variables assigned as materials would
*        need to be:
*           mesv_desc[0] = "Escalar_1"
*           mesv_desc[1] = "Escalar_9"
*
*        Then corresponding per element lists would be:
*           for list "Escalar_1"
*           escal1_lis[0] = 1    {elem scalar value for elem 1}
*           escal1_lis[1] = 1    {elem scalar value for elem 2}
*           escal1_lis[2] = 1    {elem scalar value for elem 3}
*           escal1_lis[3] = .875 {elem scalar value for elem 4}
*           escal1_lis[4] = 1    {elem scalar value for elem 5}
*           escal1_lis[5] = 1    {elem scalar value for elem 6}
*           escal1_lis[6] = .125 {elem scalar value for elem 7}
*           escal1_lis[7] = .875 {elem scalar value for elem 8}
```

```
 *           escal1_lis[8] = 1    {elem scalar value for elem 9}
 *
 *           for list "Escalar_9"
 *           escal1_lis[0] = 0    {elem scalar value for elem 1}
 *           escal1_lis[1] = 0    {elem scalar value for elem 2}
 *           escal1_lis[2] = 0    {elem scalar value for elem 3}
 *           escal1_lis[3] = .125 {elem scalar value for elem 4}
 *           escal1_lis[4] = 0    {elem scalar value for elem 5}
 *           escal1_lis[5] = 0    {elem scalar value for elem 6}
 *           escal1_lis[6] = .875 {elem scalar value for elem 7}
 *           escal1_lis[7] = .125 {elem scalar value for elem 8}
 *           escal1_lis[8] = 0    {elem scalar value for elem 9}
 *
 * Note3: Only Note1 or Note2 can be used; not both simultaneously.
 *        Note1 should be used for readers that accommodate sparse material lists.
 *        Note2 should be used for readers that have the materials defined as
 *        per element scalar variables.
 *-----------------------------------------------------------------------*/
int
USERD_get_number_of_material_sets( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_materials
 *                                              (version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Get the number of materials in the material set
 *
 *  (IN) set_index             = the material set index (zero based)
 *
 *  returns: Num_materials[set_index] = number of materials in set
 *                                     (Zero would indicate that you have
 *                                      no materials to deal with in the
 *                                      material set)
 *               or
 *
 *           -1 if an error condition
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *  * Will not be called if Num_material_sets is zero
 *  * You may want to keep this as a global for use in other routines.
 *  * This function only works when USERD_get_matf_set_type returns
 *    Z_MISET_VIA_SPARSE_MIX; and thus, is NOT available if USERD_get_matf_set_type
 *    returns Z_MISET_VIA_ESCAL_VARS.
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_materials( int set_index )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_model_parts
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the total number of unstructured and structured parts
 *   in the model, for which you can supply information.
 *
 *   This value is typically called:  Numparts_available
 *
 *  returns:  Numparts_available  (>0 if okay, <=0 if probs)
 *
 *  Notes:
 *  * IMPORTANT!!  The part or block numbers that get passed to various
 *                 routines in this API, will be the one-based table index
 *                 of these parts.
 *
 *                 For example, if you have three parts, the part or block
 *                 numbers of these parts will be:  1,2,3
 *
 *  * If going to have to read down through the parts in order to
 *    know how many, you may want to build a table of pointers to
 *    the various parts, so can easily get to particular parts in
 *    later processes.  If you can simply read the number of parts
 *    at the head of the file, then you would probably not build the
 *    table at this time.
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_model_parts( void )
```

```
/*-------------------------------------------------------------------
 USERD_get_number_of_species
 *                                          (version 2.05 and later)
 *-------------------------------------------------------------------
 *
 *   Get the number of material species in the material set
 *
 *  (IN)  set_index           = the material set index (zero based)
 *
 *  returns: Num_species[set_index] = number of material species in set
 *                                   (Zero would indicate that you have
 *                                    no materials to deal with in the
 *                                    material set)
 *               or
 *
 *           -1 if an error condition
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *  * Will not be called if Num_material_sets is zero
 *  * You may want to keep this as a global for use in other routines.
 *  * This function only works when USERD_get_matf_set_type returns
 *    Z_MISET_VIA_SPARSE_MIX; and is not available if USERD_get_matf_set_type
 *    returns Z_MISET_VIA_ESCAL_VARS
 *-------------------------------------------------------------------*/
int
USERD_get_number_of_species( int set_index )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_timesets
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the number of timesets used in the model.
 *
 *  returns:
 *  If you have a static model, both geometry and variables, you should
 *  return a value of zero.
 *
 *  If you have a transient model, then you should return one or more.
 *
 *  For example:
 *
 *     Geometry    Variables                                  No. of timesets
 *     ---------   ------------------------------             ---------------
 *     static      static                                       0
 *     static      transient, all using same timeset            1
 *
 *     transient   transient, all using same timeset as geom    1
 *
 *     static      transient, using 3 different timesets        3
 *
 *     transient   transient, using 3 different timesets and
 *                            none of them the same as the
 *                            geometry timeset                  4
 *         etc.
 *
 *  NOTE: ALL GEOMETRY MUST USE THE SAME TIMESET!!! You will have to provide
 *                                                  the timeset number to use
 *                                                  for geometry in:
 *                                            USERD_get_geom_timeset_number
 *
 *        Variables can use the same timeset as the geometry, or can use
 *        other timesets. More than one variable can use the same timeset.
 *
 *  example:  changing geometry at 5 steps, 0.0, 1.0, 2.0, 3.0, 4.0
 *            variable 1 provided at these same five steps
 *            variable 2 provided at 3 steps, 0.5, 1.25, 3.33
 *
 *       This routine should return a value of 2, because only
 *       two different timesets are needed. Timeset 1 would be for the
 *       geometry and variable 1 (they both use it). Timeset 2 would
 *       be for variable 2, which needs its own in this case.
 *
 *  Notes:
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_timesets( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_number_of_variables
 *                                              (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the number of variables (includes constant, scalar,
 *   vector and tensor types), for which you will be providing info.
 *
 *  returns: number of variables (includes constant, scalar, vector,
 *            and tensor types)
 *            >=0 if okay
 *            <0 if problem
 *
 *  Notes:
 *   ************************************************************
 *  * Variable numbers, by which references will be made, are implied
 *    here. If you say there are 3 variables, the variable numbers
 *    will be 1, 2, and 3.
 *   ************************************************************
 *
 *  * Num_variables would be set here
 *----------------------------------------------------------------------*/
int
USERD_get_number_of_variables( void )
```

```
/*----------------------------------------------------------------------
 USERD_get_part_coords
 *                                          (version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the coordinates for an unstructured part.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) coord_array             = 2D float array which contains
 *                                  x,y,z coordinates of each node.
 *
 *       (IMPORTANT: the second dimension of of this array is 1-based!!!)
 *
 *                                  (Array will have been allocated
 *                                   3 by (number_of_nodes + 1) for the part
 *                                   long - see USERD_get_gold_part_build_info)
 *
 *                        ex) If number_of_nodes = 100
 *                            as obtained in:
 *                              USERD_get_gold_part_build_info
 *
 *                            Then the allocated dimensions of the
 *                            pointer sent to this routine will be:
 *                              coord_array[3][101]
 *
 *                            Ignore the coord_array[0][0]
 *                                       coord_array[1][0]
 *                                       coord_array[2][0] locations and start
 *                            the node coordinates at:
 *                              coord_array[0][1]
 *                              coord_array[1][1]
 *                              coord_array[2][1]
 *
 *                              coord_array[0][2]
 *                              coord_array[1][2]
 *                              coord_array[2][2]
 *
 *                                    etc.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *  * Not called unless Num_unstructured_parts is > 0
 *  * Not called unless number_of_nodes for the part > 0
 *----------------------------------------------------------------------*/
int
USERD_get_part_coords(int part_number,
                      float **coord_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_coords_in_buffers
 *                                <optional>  (version 2.08 and later)
 *----------------------------------------------------------------------
 *   Get the coordinates for an unstructured part in buffers.
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines) in
 *   your reader (in a proper manner), you should be able to not only get the
 *   execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_part_coords routine. And thus its implementation should
 *   not be too difficult to add to any existing reader that has already
 *   implemented the USERD_get_part_coords routine.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  first                   = TRUE if first invocation of a buffered set.
 *                                  Will be FALSE for all subsequent invocations
 *                                  of the set.  This is so you can open files,
 *                                  get to the correct starting spot,
 *                                  initialize, etc.
 *
 *  (IN) n_beg                    = Zero based, first node index
 *                                  of the buffered set
 *
 *  (IN) n_end                    = Zero based, last node index
 *                                  of the buffered set
 *
 *                     Thus, for first five nodes:
 *                       n_beg = 0
 *                       n_end = 4
 *                       total_number = (n_end - n_beg) + 1 = (4 - 0) + 1 = 5
 *
 *                     for second five nodes, would be:
 *                       n_beg = 5
 *                       n_end = 9
 *                       total_number = (n_end - n_beg) + 1 = (9 - 5) + 1 = 5
 *
```

```
*                     for all nodes of a part, would be:
*                       n_beg = 0
*                       n_end = num_nodes - 1
*
*  (IN)  buffer_size             = The size of the buffer.
*                                  Namely:   coord_array[3][buffer_size]
*
*  (OUT) coord_array             = 2D float buffer array which is set up to
*                                  hold x,y,z coordinates of nodes.
*
*       (IMPORTANT: the second dimension of of this array is 0-based!!!)
*
*       (IMPORTANT: in the sister routine (USERD_get_part_coords) - which
*                   does not use buffers. This array is 1-based.
*                   So pay attention.)
*
*                                   (Array will have been allocated
*                                    3 by buffer_size long
*
* Example, if we had a part with 645 nodes and the buffer size was set to 200
*
*  first invocation:
*    first = TRUE            Will be TRUE the first time!
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    coord_array[3][200]     fill with values for nodes 1 - 200 (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  second invocation:        which occurs because we returned a 0 last time
*    first = FALSE           will now be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    coord_array[3][200]     fill with values for nodes 201 - 400 (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  third invocation:         which occurs because we returned a 0 last time
*    first = FALSE           will still be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    coord_array[3][200]     fill with values for nodes 401 - 600 (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  fourth invocation:        which occurs because we returned a 0 last time
*    first = FALSE           will still be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    coord_array[3][200]     fill with values for nodes 601 - 645 (zero-based)
*    *num_returned = 45      set this
*    return(1)               return this (indicates done!)
*
*  (OUT)  *num_returned  = The number of nodes whose coordinates are returned
*                          in the buffer. This will normally be equal to
*                          buffer_size except for that last buffer -
*                          which could be less than a full buffer.
```

```
 *
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless number_of_nodes for the part > 0
 *
 *  * Again, make sure each buffer is zero based. For our example above:
 *
 *                                                Invocation:
 *                                     1          2          3          4
 *                                  -------    -------    --------   -------
 *      coord_array[0][0]    x  for node 1     node 201   node 401   node 601
 *      coord_array[1][0]    y  for    "          "          "          "
 *      coord_array[2][0]    z  for    "          "          "          "
 *
 *      coord_array[0][1]    x  for node 2     node 202   node 402   node 602
 *      coord_array[1][1]    y  for    "          "          "          "
 *      coord_array[2][1]    z  for    "          "          "          "
 *
 *      ...
 *
 *      coord_array[0][199]  x  for node 200   node 400   node 600   node 645
 *      coord_array[1][199]  y  for    "          "          "          "
 *      coord_array[2][199]  z  for    "          "          "          "
 *-----------------------------------------------------------------------*/
int
USERD_get_part_coords_in_buffers(int part_number,
                                 float **coord_array,
                                 int first,
                                 int n_beg,
                                 int n_end,
                                 int buffer_size,
                                 int *num_returned)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_element_ids_by_type
 *                                          (version 2.00 and later)
 *                                (Modified at 2.01 as described below)
 *                                (Modified at 2.03 as described below)
 *----------------------------------------------------------------------
 *
 *   Prior to API 2.01:
 *   ==================
 *   Gets the ids for the elements of a particular type for an
 *   unstructured part.
 *
 *   Starting at API 2.01:
 *   ====================
 *   Gets the ids for the elements of a particular type for an
 *   unstructured or structured part.
 *
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  element_type            = One of the following (See global_extern.h)
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *                                  Z_TET04    4 node tetrahedron
 *                                  Z_TET10   10 node tetrahedron
 *                                  Z_PYR05    5 node pyramid
 *                                  Z_PYR13   13 node pyramid
 *                                  Z_PEN06    6 node pentahedron
 *                                  Z_PEN15   15 node pentahedron
 *                                  Z_HEX08    8 node hexahedron
 *                                  Z_HEX20   20 node hexahedron
 *   Starting at API 2.03           Z_NSIDED  nsided polygon
 *   Starting at API 2.03           Z_NFACED  nfaced polyhedron
 *
 *   Starting at API 2.01:
 *   ====================
 *                                  Z_G_POINT    ghost node point element
 *                                  Z_G_BAR02    2 node ghost bar
 *                                  Z_G_BAR03    3 node ghost bar
 *                                  Z_G_TRI03    3 node ghost triangle
 *                                  Z_G_TRI06    6 node ghost triangle
 *                                  Z_G_QUA04    4 node ghost quad
 *                                  Z_G_QUA08    8 node ghost quad
 *                                  Z_G_TET04    4 node ghost tetrahedron
 *                                  Z_G_TET10   10 node ghost tetrahedron
 *                                  Z_G_PYR05    5 node ghost pyramid
 *                                  Z_G_PYR13   13 node ghost pyramid
 *                                  Z_G_PEN06    6 node ghost pentahedron
 *                                  Z_G_PEN15   15 node ghost pentahedron
```

```
 *                                  Z_G_HEX08    8 node ghost hexahedron
 *                                  Z_G_HEX20   20 node ghost hexahedron
 *   Starting at API 2.03           Z_G_NSIDED  ghost nsided polygon
 *   Starting at API 2.03           Z_G_NFACED  ghost nfaced polyhedron
 *
 *  (OUT) elemid_array            = 1D array containing id of each
 *                                    element of the type.
 *
 *                                  (Array will have been allocated
 *                                   number_of_elements of type long)
 *
 *                        ex) If number_of_elements[Z_TRI03] = 25
 *                               number_of_elements[Z_QUA04] = 100
 *                               number_of_elements[Z_HEX08] = 30
 *                            as obtained in:
 *                             USERD_get_gold_part_build_info
 *
 *                            Then the allocated dimensions available
 *                            for this routine will be:
 *                               elemid_array[25]    when called with Z_TRI03
 *
 *                               elemid_array[100]   when called with Z_QUA04
 *
 *                               elemif_array[30]    when called with Z_HEX08
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  Prior to API 2.01:
 *  ==================
 *  * Not called unless Num_unstructured_parts is > 0  and element
 *    label status is TRUE
 *
 *  Starting at API 2.01:
 *  =====================
 *  * Not called unless element label status is TRUE in
 *    USERD_get_element_label_status
 *-----------------------------------------------------------------------*/
int
USERD_get_part_element_ids_by_type(int part_number,
                                   int element_type,
                                   int *elemid_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_element_ids_by_type_in_buffers
 *                                <optional>  (version 2.08 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the ids for the elements of a particular type
 *   in an unstructured part in buffers
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will
 *   require much more memory on each server to read in the data (somewhat
 *   like each server having to read the whole model). You will however, get
 *   the execution advantage of having your model partitioned across multiple
 *   servers.
 *
 *   If you do implement this routine (and the other in_buffers routines)
 *   in your reader (in a proper manner), you should be able to not only get
 *   the execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_part_element_ids_by_type routine. And thus its
 *   implementation should not be too difficult to add to any existing reader
 *   that has already implemented the USERD_get_part_element_ids_by_type
 *   routine.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  element_type            = One of the following (See global_extern.h)
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *                                  Z_TET04    4 node tetrahedron
 *                                  Z_TET10   10 node tetrahedron
 *                                  Z_PYR05    5 node pyramid
 *                                  Z_PYR13   13 node pyramid
 *                                  Z_PEN06    6 node pentahedron
 *                                  Z_PEN15   15 node pentahedron
 *                                  Z_HEX08    8 node hexahedron
 *                                  Z_HEX20   20 node hexahedron
 *
 *
```

```
*   Starting at API 2.01:
*   =====================
*                                  Z_G_POINT    ghost node point element
*                                  Z_G_BAR02    2 node ghost bar
*                                  Z_G_BAR03    3 node ghost bar
*                                  Z_G_TRI03    3 node ghost triangle
*                                  Z_G_TRI06    6 node ghost triangle
*                                  Z_G_QUA04    4 node ghost quad
*                                  Z_G_QUA08    8 node ghost quad
*                                  Z_G_TET04    4 node ghost tetrahedron
*                                  Z_G_TET10   10 node ghost tetrahedron
*                                  Z_G_PYR05    5 node ghost pyramid
*                                  Z_G_PYR13   13 node ghost pyramid
*                                  Z_G_PEN06    6 node ghost pentahedron
*                                  Z_G_PEN15   15 node ghost pentahedron
*                                  Z_G_HEX08    8 node ghost hexahedron
*                                  Z_G_HEX20   20 node ghost hexahedron
*                                  Z_NSIDED     n node ghost nsided polygon
*                                  Z_NFACED     n face ghost nfaced polyhedron
*
*   Starting at API 2.02:
*   =====================
*                                  Z_NSIDED     n node nsided polygon
*                                  Z_NFACED     n face nfaced polyhedron
*                                  Z_G_NSIDED   n node ghost nsided polygon
*                                  Z_G_NFACED   n face ghost nfaced polyhedron
*
*  (IN)  first                   = TRUE if first invocation of a buffered set.
*                                  Will be FALSE for all subsequent invocations
*                                  of the set.  This is so you can open files,
*                                  get to the correct starting spot,
*                                  initialize, etc.
*
*  (IN) e_beg                    = Zero based, first element number
*                                  of the buffered set
*
*  (IN) e_end                    = Zero based, last element number
*                                  of the buffered set
*
*                     Thus, for first five elements of a type:
*                       e_beg = 0
*                       e_end = 4
*                       total_number = (e_end - e_beg) + 1 = (4 - 0) + 1 = 5
*
*                     for second five elements of a type, would be:
*                       e_beg = 5
*                       e_end = 9
*                       total_number = (e_end - e_beg) + 1 = (9 - 5) + 1 = 5
*
*                     for all elements of the type of a part, would be:
*                       n_beg = 0
*                       n_end = num_elements_of_type - 1
*
*  (IN) buffer_size              = The size of the buffer.
*                                  Namely:  elemid_array[buffer_size]
*
*  (OUT) elemid_array            = 1D buffer array which is set up to hold ids
*                                  of elements of the type.
*
*                                  (Array will have been allocated
*                                   buffer_size long)
```

```
*
*   * Example, (if 158 quad elements, and buffer size is 200)
*
*       (get all 158 quad4 ids in one invocation)
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0                 (zero based, first element index)
*         e_end  = 157               (zero based, last element index)
*         buffer_size = 200
*         elemeid_array[200]         Use first 158 locations of the array
*         *num_returned = 158        set this
*         return(1)                  return this (indicates no more to do)
*
*   * Example, (if 158 quad elements, and buffer size is 75)
*
*        first invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         elemid_array[75]           load in ids for elements 1 - 75
*         *num_returned = 75         set this
*         return(0)                  return this (indicates more to do)
*
*        second invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         elemid_array[75]           load in ids for elements 76 - 150
*         *num_returned = 75         set this
*         return(0)                  return this (indicates more to do)
*
*        third invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         elemid_array[75]           load in ids for elements 151 - 158
*         *num_returned = 8          set this
*         return(1)                  return this (indicates no more to do)
*
*
*  (OUT)  *num_returned       = The number of elements whose ids are returned
*                               in the buffer. This will normally be equal
*                               to buffer_size except for that last buffer
*                               - which could be less than a full buffer.
*
* returns  0  if got some, more to do
*          1  if got some, done
*         -1  if an error
*
*  Notes:
*  * This will be based on Current_time_step
*
*  * Again, make sure each buffer is zero based.
*    For our example using buffers above:
*
```

```
 *                                                 Invocation:
 *                                          1          2          3
 *                                       -------    -------    --------
 *      elemid_array[0]       elem id  for quad 1    quad 76     quad 151
 *
 *      elemid_array[1]       elem id  for quad 2    quad 77     quad 152
 *
 *      ...
 *
 *      elemid_array[74]      elem id  for quad 75   quad 150    quad 158
 *-----------------------------------------------------------------------*/
int
USERD_get_part_element_ids_by_type_in_buffers(int part_number,
                                              int element_type,
                                              int *elemid_array,
                                              int first,
                                              int e_beg,
                                              int e_end,
                                              int buffer_size,
                                              int *num_returned)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_elements_by_type
 *                                          (version 2.00 and later)
 *                               (Modified at 2.01 as described below)
 *                               (Modified at 2.03 as described below)
 *----------------------------------------------------------------------
 *
 *   Gets the connectivities for the elements of a particular type
 *   in an unstructured part
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                  It is NOT the part_id that is
 *                                  loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  element_type            = One of the following (See global_extern.h)
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *                                  Z_TET04    4 node tetrahedron
 *                                  Z_TET10   10 node tetrahedron
 *                                  Z_PYR05    5 node pyramid
 *                                  Z_PYR13   13 node pyramid
 *                                  Z_PEN06    6 node pentahedron
 *                                  Z_PEN15   15 node pentahedron
 *                                  Z_HEX08    8 node hexahedron
 *                                  Z_HEX20   20 node hexahedron
 *   Starting at API 2.03           Z_NSIDED  nsided polygon
 *   Starting at API 2.03           Z_NFACED  nfaced polyhedron
 *
 *   Starting at API 2.01:
 *   ====================
 *                                  Z_G_POINT    ghost node point element
 *                                  Z_G_BAR02    2 node ghost bar
 *                                  Z_G_BAR03    3 node ghost bar
 *                                  Z_G_TRI03    3 node ghost triangle
 *                                  Z_G_TRI06    6 node ghost triangle
 *                                  Z_G_QUA04    4 node ghost quad
 *                                  Z_G_QUA08    8 node ghost quad
 *                                  Z_G_TET04    4 node ghost tetrahedron
 *                                  Z_G_TET10   10 node ghost tetrahedron
 *                                  Z_G_PYR05    5 node ghost pyramid
 *                                  Z_G_PYR13   13 node ghost pyramid
 *                                  Z_G_PEN06    6 node ghost pentahedron
 *                                  Z_G_PEN15   15 node ghost pentahedron
 *                                  Z_G_HEX08    8 node ghost hexahedron
 *                                  Z_G_HEX20   20 node ghost hexahedron
 *   Starting at API 2.03           Z_G_NSIDED  ghost nsided polygon
 *   Starting at API 2.03           Z_G_NFACED  ghost nfaced polyhedron
 *
 *
 *
 *
```

```
 *  (OUT) conn_array              = 2D array containing connectivity
 *                                   of each element of the type.
 *
 *                                 (Array will have been allocated
 *                                  number_of_elements of
 *                                  the type by connectivity length
 *                                  of the type)
 *
 *                       ex) If number_of_elements[Z_TRI03] = 25
 *                              number_of_elements[Z_QUA04] = 100
 *                              number_of_elements[Z_HEX08] = 30
 *                           as obtained in:
 *                            USERD_get_gold_part_build_info
 *
 *                           Then the allocated dimensions available
 *                           for this routine will be:
 *                              conn_array[25][3]   when called with Z_TRI03
 *
 *                              conn_array[100][4]  when called with Z_QUA04
 *
 *                              conn_array[30][8]   when called with Z_HEX08
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *  * Not called unless Num_unstructured_parts is > 0
 *-------------------------------------------------------------------------*/
int
USERD_get_part_elements_by_type(int part_number,
                                int element_type,
                                int **conn_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_elements_by_type_in_buffers
 *                               <optional>  (Version 2.08 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the connectivities for the elements of a particular type
 *   in an unstructured part in buffers
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines) in
 *   your reader (in a proper manner), you should be able to not only get the
 *   execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_part_elements_by_type routine. And thus its
 *   implementation should not be too difficult to add to any existing reader
 *   that has already implemented the USERD_get_part_elements_by_type routine.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  element_type            = One of the following (See global_extern.h)
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *                                  Z_TET04    4 node tetrahedron
 *                                  Z_TET10   10 node tetrahedron
 *                                  Z_PYR05    5 node pyramid
 *                                  Z_PYR13   13 node pyramid
 *                                  Z_PEN06    6 node pentahedron
 *                                  Z_PEN15   15 node pentahedron
 *                                  Z_HEX08    8 node hexahedron
 *                                  Z_HEX20   20 node hexahedron
 *
 *
 *
 *
```

```
*   Starting at API 2.01:
*   =====================
*                                  Z_G_POINT    ghost node point element
*                                  Z_G_BAR02    2 node ghost bar
*                                  Z_G_BAR03    3 node ghost bar
*                                  Z_G_TRI03    3 node ghost triangle
*                                  Z_G_TRI06    6 node ghost triangle
*                                  Z_G_QUA04    4 node ghost quad
*                                  Z_G_QUA08    8 node ghost quad
*                                  Z_G_TET04    4 node ghost tetrahedron
*                                  Z_G_TET10   10 node ghost tetrahedron
*                                  Z_G_PYR05    5 node ghost pyramid
*                                  Z_G_PYR13   13 node ghost pyramid
*                                  Z_G_PEN06    6 node ghost pentahedron
*                                  Z_G_PEN15   15 node ghost pentahedron
*                                  Z_G_HEX08    8 node ghost hexahedron
*                                  Z_G_HEX20   20 node ghost hexahedron
*                                  Z_NSIDED     n node ghost nsided polygon
*                                  Z_NFACED     n face ghost nfaced polyhedron
*
*   Starting at API 2.02:
*   =====================
*                                  Z_NSIDED     n node nsided polygon
*                                  Z_NFACED     n face nfaced polyhedron
*                                  Z_G_NSIDED   n node ghost nsided polygon
*                                  Z_G_NFACED   n face ghost nfaced polyhedron
*
*  (IN)  first                    = TRUE if first invocation of a buffered set.
*                                   Will be FALSE for all subsequent invocations
*                                   of the set.  This is so you can open files,
*                                   get to the correct starting spot,
*                                   initialize, etc.
*
*  (IN) e_beg                     = Zero based, first element number of the buffered set
*
*  (IN) e_end                      = Zero based, last element number of the buffered set
*
*                     Thus, for first five elements of a type:
*                       e_beg = 0
*                       e_end = 4
*                       total_number = (e_end - e_beg) + 1 = (4 - 0) + 1 = 5
*
*                     for second five elements of a type, would be:
*                       e_beg = 5
*                       e_end = 9
*                       total_number = (e_end - e_beg) + 1 = (9 - 5) + 1 = 5
*
*                     for all elements of the type of a part, would be:
*                       n_beg = 0
*                       n_end = num_elements_of_type - 1
*
*  (IN) buffer_size             = The size of the buffer.
*                                 Namely:  conn_array[buffer_size][element_size]
*
*  (OUT) conn_array             = 2D buffer array which is set up to hold
*                                 connectivity of elements of the type.
*
*                                    (Array will have been allocated
*                                     buffer_size of the type by connectivity length
*                                     of the type)
*
```

```
*                  ex) The allocated dimensions available
*                      for this routine will be:
*                         conn_array[buffer_size][3]   when called with Z_TRI03
*
*                         conn_array[buffer_size][4]   when called with Z_QUA04
*
*                         conn_array[buffer_size][8]   when called with Z_HEX08
*
*                         etc.
*
*   * Example, (if 158 quad elements, and buffer size is 200)
*
*       (get all 158 quad4s in one invocation)
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0                 (zero based, first element index)
*         e_end  = 157               (zero based, last element index)
*         buffer_size = 200
*         conn_array[200][4]         Use first 158 locations of the array
*         *num_returned = 158        set this
*         return(1)                  return this (indicates no more to do)
*
*   * Example, (if 158 quad elements, and buffer size is 75)
*
*        first invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         conn_array[75][4]          load in conn for elements 1 - 75
*         *num_returned = 75         set this
*         return(0)                  return this (indicates more to do)
*
*        second invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         conn_array[75][4]          load in conn for elements 76 - 150
*         *num_returned = 75         set this
*         return(0)                  return this (indicates more to do)
*
*        third invocation:
*         element_type = Z_QUA04
*         first = TRUE               Will be TRUE the first time!
*         e_beg  = 0
*         e_end  = 157
*         buffer_size = 75
*         conn_array[75][4]          load in conn for elements 151 - 158
*         *num_returned = 8          set this
*         return(1)                  return this (indicates no more to do)
*
*
*  (OUT)  *num_returned          = The number of elements whose connectivities
*                                  are returned in the buffer. This will
*                                  normally be equal to buffer_size except for
*                                  that last buffer - which could be less than
*                                  a full buffer.
*
```

```
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Again, make sure each buffer is zero based.
 *    For our example using buffers above:
 *
 *                                                    Invocation:
 *                                                1          2          3
 *                                             -------    -------    --------
 *      conn_array[0][0]       node 1 in conn for quad 1    quad 76     quad 151
 *      conn_array[0][1]       node 2 in conn for quad 1    quad 76     quad 151
 *      conn_array[0][2]       node 3 in conn for quad 1    quad 76     quad 151
 *      conn_array[0][3]       node 4 in conn for quad 1    quad 76     quad 151
 *
 *      conn_array[1][0]       node 1 in conn for quad 2    quad 77     quad 152
 *      conn_array[1][1]       node 2 in conn for quad 2    quad 77     quad 152
 *      conn_array[1][2]       node 3 in conn for quad 2    quad 77     quad 152
 *      conn_array[1][3]       node 4 in conn for quad 2    quad 77     quad 152
 *
 *      ...
 *
 *      conn_array[74][0]      node 1 in conn for quad 75    quad 150   quad 158
 *      conn_array[74][1]      node 2 in conn for quad 75    quad 150   quad 158
 *      conn_array[74][2]      node 3 in conn for quad 75    quad 150   quad 158
 *      conn_array[74][3]      node 4 in conn for quad 75    quad 150   quad 158
 *-------------------------------------------------------------------------*/
int
USERD_get_part_elements_by_type_in_buffers(int part_number,
                                           int element_type,
                                           int **conn_array,
                                           int first,
                                           int e_beg,
                                           int e_end,
                                           int buffer_size,
                                           int *num_returned)
```

```
/*----------------------------------------------------------------------
  USERD_get_part_node_ids
 *                                          (Version 2.00 and later)
 *                                (Modified at 2.01 as described below)
 *----------------------------------------------------------------------
 *   Prior to API 2.01:
 *   ==================
 *   Get the node ids of an unstructured part.
 *
 *   Starting at API 2.01:
 *   ====================
 *   Get the node ids of an unstructured or structured part.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) nodeid_array            = 1D array containing node ids of
 *                                   each node in the part.
 *
 *       (IMPORTANT: this array is 1-based!!!)
 *
 *                                  (Array will have been allocated
 *                                   (number_of_nodes + 1) for the part long
 *                                   see USERD_get_gold_part_build_info)
 *
 *                       ex) If number_of_nodes = 100
 *                           as obtained in:
 *                             USERD_get_gold_part_build_info
 *
 *                           Then the allocated dimensions of the
 *                           pointer sent to this routine will be:
 *                             nodeid_array[101]
 *
 *                           Ignore the nodeid_array[0] location and start
 *                           the node ids at:
 *                             nodeid_array[1]
 *
 *                             nodeid_array[2]
 *
 *                                   etc.
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless number_of_nodes for the part is > 0  and
 *    node label status is TRUE, as returned from USERD_get_node_label_status
 *
 * * The ids are purely labels, used when displaying or querying node ids.
 *   However, any node id < 0 will never be displayed
 *----------------------------------------------------------------------*/
int
USERD_get_part_node_ids(int part_number,
                        int *nodeid_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_part_node_ids_in_buffers
 *                                  <optional>  (Version 2.08 an later)
 *----------------------------------------------------------------------
 *
 *   Get the node ids for an unstructured part in buffers.
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines)
 *   in your reader (in a proper manner), you should be able to not only get
 *   the execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_part_node_ids routine. And thus its implementation should
 *   not be too difficult to add to any existing reader that has already
 *   implemented the USERD_get_part_node_ids routine.
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that is
 *                            loaded in USERD_get_gold_part_build_info)
 *
 *  (IN)  first            = TRUE if first invocation of a buffered set.
 *                           Will be FALSE for all subsequent invocations
 *                           of the set.  This is so you can open files, get to
 *                           the correct starting spot, initialize, etc.
 *
 *  (IN) n_beg             = Zero based, first node index
 *                           of the buffered set
 *
 *  (IN) n_end             = Zero based, last node index
 *                           of the buffered set
 *
 *                     Thus, for first five nodes:
 *                       n_beg = 0
 *                       n_end = 4
 *                       total_number = (n_end - n_beg) + 1 = (4 - 0) + 1 = 5
 *
 *                     for second five nodes, would be:
 *                       n_beg = 5
 *                       n_end = 9
 *                       total_number = (n_end - n_beg) + 1 = (9 - 5) + 1 = 5
 *
```

```
*                     for all nodes of a part, would be:
*                       n_beg = 0
*                       n_end = num_nodes - 1
*
*  (IN)  buffer_size             = The size of the buffer.
*                                  Namely:   nodeid_array[buffer_size]
*
*  (OUT) nodeid_array            = 1D buffer array which is set up to hold
*                                  node ids of nodes
*
*       (IMPORTANT: this array is 0-based!!!)
*
*       (IMPORTANT: in the sister routine (USERD_get_part_node_ids) - which
*                   does not use buffers. This array is 1-based.
*                   So pay attention.)
*
*                                   (Array will have been allocated
*                                    buffer_size long)
*
* Example, if we had a part with 645 nodes and the buffer size was set to 200
*
*  first invocation:
*    first = TRUE            Will be TRUE the first time!
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    nodeid_array[200]       fill with values for nodes 1 - 200   (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  second invocation:        which occurs because we returned a 0 last time
*    first = FALSE           will now be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    nodeid_array[200]       fill with values for nodes 201 - 400 (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  third invocation:         which occurs because we returned a 0 last time
*    first = FALSE           will still be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    nodeid_array[200]       fill with values for nodes 401 - 600 (zero-based)
*    *num_returned = 200     set this
*    return(0)               return this (indicates more to do)
*
*  fourth invocation:        which occurs because we returned a 0 last time
*    first = FALSE           will still be FALSE
*    n_beg = 0
*    n_end = 644
*    buffer_size = 200
*    nodeid_array[200]       fill with values for nodes 601 - 645 (zero-based)
*    *num_returned = 45      set this
*    return(1)               return this (indicates done!)
*
*  (OUT)  *num_returned          = The number of nodes whose ids are returned
*                                  in the buffer. This will normally be equal
*                                  to buffer_size except for that last buffer
*                                  - which could be less than a full buffer.
```

```
 *
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless number_of_nodes for the part > 0
 *
 *  * Again, make sure each buffer is zero based. For our example above:
 *
 *                                                Invocation:
 *                                     1          2          3          4
 *                                  -------    -------    --------    -------
 *      nodeid_array[0]    id   for node 1     node 201   node 401    node 601
 *
 *      nodeid_array[1]    id   for node 2     node 202   node 402    node 602
 *
 *      ...
 *
 *      nodeid_array[199]  id   for node 200   node 400   node 600    node 645
 *-------------------------------------------------------------------------*/
int
USERD_get_part_node_ids_in_buffers(int part_number,
                                   int *nodeid_array,
                                   int first,
                                   int n_beg,
                                   int n_end,
                                   int buffer_size,
                                   int *num_returned)
```

```
/*-----------------------------------------------------------------------
 USERD_get_periodic_ghosts_num_pairs
 *                                <optional>   (Version 2.00 and later)
  *----------------------------------------------------------------------
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *         face            = The (zero-based) symmetry face number. This routine will
 *                           be called the number of times as the number of faces
 *                           declared in USERD_get_periodic_ghosts_num_symmety_faces.
 *
 *                           For translation and rotation, this routine will be
 *                             called twice - once with face = 0, and once with
 *                             face = 1;
 *                           For mirroring, this routine can be called once,
 *                             twice, or three times. Thus face will be 0, 0 and 1,
 *                             or 0,1,2.
 *
 *  returns: the number of node pairs for the symmetry face.
 *
 *  Notes:
 *   You must have first called:
 *      USERD_use_periodic_ghosts   (only need be done once for the model)
 *      USERD_get_periodic_ghosts_num_symmetry_faces
 *
 *   and will need to call:
 *      USERD_get_periodic_ghosts_pairs
 *
 *   See USERD_use_periodic_ghosts routine for examples.
 *----------------------------------------------------------------------*/
int
USERD_get_periodic_ghosts_num_pairs(int part_number, int face)
```

```
/*----------------------------------------------------------------------
 USERD_get_periodic_ghosts_num_symmetry_faces
 *                                <optional>   (Version 2.00 and later)
  *--------------------------------------------------------------------
 *
 *  (IN)  part_number      = The part number
 *
 *                           (1-based index of part table, namely:
 *
 *                              1 ... Numparts_available.
 *
 *                            It is NOT the part_id that
 *                            is loaded in USERD_get_gold_part_build_info)
 *
 *  returns: The number of symmetry faces in the part, and for which the
 *           pairs of matching nodes will be provided in:
 *           USERD_get_periodic_ghosts_num_pairs
 *           USERD_get_periodic_ghosts_pairs.
 *
 *           or the ijk information will be provided in:
 *           USERD_get_periodic_ghosts_structured_face_info
 *
 *           Zero should be returned if this part has no symmetry faces.
 *
 *  Notes:
 *   You must have first called:
 *      USERD_use_periodic_ghosts   (only need be done once for the model)
 *
 *   and will afterwards need to call either:
 *      USERD_get_periodic_ghosts_num_pairs    and
 *      USERD_get_periodic_ghosts_pairs
 *
 *   or
 *      USERD_get_periodic_ghosts_structured_face_info
 *
 *
 *   See USERD_use_periodic_ghosts routine for examples.
 *
 *   For Translation, and rotation, the number should be 2 because there
 *    should always be an opposite face.
 *
 *   For mirroring, the number can be 1, 2, or 3, depending on which of
 *    the reflection faces are being used.  Namely, for mirror symmetry
 *    the original instance can reflect across x, y, z, xy, yz, xz, or xyz
 *    filling the appropriate other 7 quadrants. And the faces needing ghosts
 *    depends on which are used. (x, xy, xz, xyz need x face,
 *                                y, xy, yz, xyz need y face,
 *                                z, xz, yz, xyz need z face)
 *--------------------------------------------------------------------*/
int
USERD_get_periodic_ghosts_num_symmetry_faces(int part_number)
```

```
/*----------------------------------------------------------------------
 USERD_get_periodic_ghosts_pairs
 *                               <optional>   (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  (IN)  part_number  = The part number
 *
 *                       (1-based index of part table, namely:
 *
 *                          1 ... Numparts_available.
 *
 *                        It is NOT the part_id that
 *                        is loaded in USERD_get_gold_part_build_info)
 *
 *         face          = The (zero-based) symmetry face number. This routine will
 *                         be called the number of times as the number of faces
 *                         declared in USERD_get_periodic_ghosts_num_symmety_faces.
 *
 *                         For translation and rotation, this routine will be
 *                           called twice - once with face = 0, and once with
 *                           face = 1;
 *                         For mirroring, this routine can be called once,
 *                           twice, or three times. Thus face will be 0, 0 and 1,
 *                           or 0,1,2.
 *
 *     providing_matrix  = TRUE if the reader is providing the matrix. And the
 *                             m argument must contain that matrix.
 *
 *                         FALSE if EnSight should compute it from the node pairs.
 *                               Thus  m argument is ignored.
 *
 *                          Note:  EnSight should be able to produce the m matrix
 *                                 for you, except for the condition of planar 2D
 *                                 with transformations out of plane.
 *
 *      m                = 4x4 transformation matrix.
 *                         This will be used if providing_matrix flag is TRUE
 *
 *                         Note: this matrix will be used in the following fashion:
 *
 *                           {Ax Ay Az 1}|m11 m12 m13 m14| = {Bx By Bz 1}
 *                               .       |m21 m22 m23 m24|       .
 *                               .       |m31 m32 m33 m34|       .
 *                               .       |m41 m42 m43 m44|       .
 *
 *                           where Ax Ay Az are the coords of node1 of a pair
 *                                 Bx By Bz are the coords of node2 of a pair
 *
 *                           This matrix is generally thought of as:
 *
 *                              |              | 0|   r is the rotational, shearing,
 *                              |      r       | 0|     reflection portion
 *                              |              | 0|
 *                              |--------------|--|
 *                              |      t       | 1|   t is the translational portion
 *
 *
 *  (OUT) pairs        = array of part node indicies that are periodic match pairs
 *                       for the part and symmetry face.
 *
 *                       pairs[0] = node1 of 1st pair
```

```
 *                             [1] = node2 of 1st pair
 *                             [2] = node1 of 2nd pair
 *                             [3] = node2 of 2nd pair
 *                             [4] = node1 of 3rd pair
 *                             [5] = node2 of 3rd pair
 *                                    .
 *                                    .
 *                                    .
 *
 *                          it's length needs to be twice the num_pairs returned in
 *                          USERD_get_periodic_ghosts_num_pairs.
 *
 *                          The nodes need to be the node indicies, not the ids.
 *
 *  returns: Z_OK    if all is well
 *           Z_ERR   if an error
 *
 *  Notes:
 *   You must have first called:
 *     USERD_use_periodic_ghosts   (only need be done once for the model)
 *     USERD_get_periodic_ghosts_num_symmetry_faces
 *     USERD_get_periodic_ghosts_num_pairs
 *
 *
 *   See USERD_use_periodic_ghosts routine for examples.
 *
 *   For translation and rotation, the pairs for face 1 will be the reverse
 *     of those for face 0.
 *   For mirroring, each pair will contain the same node index.
 *-----------------------------------------------------------------------*/
int
USERD_get_periodic_ghosts_pairs(int part_number,
                                int face,
                                int *providing_matrix,
                                double m[4][4],
                                int *pairs)
```

```
/*----------------------------------------------------------------------
 USERD_get_periodic_ghosts_structured_face_info
 *                               <optional>   (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  (IN)  part_number  = The part number
 *
 *                       (1-based index of part table, namely:
 *
 *                          1 ... Numparts_available.
 *
 *                        It is NOT the part_id that
 *                        is loaded in USERD_get_gold_part_build_info)
 *
 *         face          = The (zero-based) symmetry face number. This routine will
 *                         be called the number of times as the number of faces
 *                         declared in USERD_get_periodic_ghosts_num_symmetry_faces.
 *
 *                         For translation and rotation, this routine will be
 *                           called twice - once with face = 0, and once with
 *                           face = 1;
 *                         For mirroring, this routine can be called once,
 *                           twice, or three times. Thus face will be 0, 0 and 1,
 *                           or 0,1,2.
 *
 *     providing_matrix  = TRUE if the reader is providing the matrix. And the
 *                              m argument must contain that matrix.
 *
 *                         FALSE if EnSight should compute it from the node pairs.
 *                               Thus  m argument is ignored.
 *
 *                          Note:  EnSight should be able to produce the m matrix
 *                                 for you.
 *                                 Note that there is an issue with the
 *                                 condition of planar 2D transformations out of
 *                                 plane. The transformation matrix produced may not
 *                                 be what you wanted - because there isn't enough
 *                                 information provided.  However, even if the
 *                                 transformation isn't exactly correct, the ghost
 *                                 elements produced may still do their job of
 *                                 proper interpolation across the boundary (unless
 *                                 direction is critical).  So for scalars you should
 *                                 be good - but for vectors you may not be.
 *
 *
 *      m                  = 4x4 transformation matrix.
 *                           This will be used if providing_matrix flag is TRUE
 *
 *                           Note: this matrix will be used in the following fashion:
 *
 *                             {Ax Ay Az 1}|m11 m12 m13 m14| = {Bx By Bz 1}
 *                                 .       |m21 m22 m23 m24|       .
 *                                 .       |m31 m32 m33 m34|       .
 *                                 .       |m41 m42 m43 m44|       .
 *
 *                             where Ax Ay Az are the coords of node1 of a pair
 *                                   Bx By Bz are the coords of node2 of a pair
 *
 *                             This matrix is generally thought of as:
 *
 *                                |              | 0|   r is the rotational, shearing,
```

```
*                              |      r       | 0|     reflection portion
*                              |              | 0|
*                              |--------------|--|
*                              |      t       | 1|   t is the translational portion
*
*
*  (OUT) which_ijk    = 1 for i direction or axis
*                       2 for j direction or axis
*                       3 ofr k direction or axis
*
*        min_or_max    = 0 for min i,j, or k
*                        1 for max i,j, or k
*
*  returns: Z_OK    if all is well
*           Z_ERR   if an error
*
*  Notes:
*  You must first call:
*    USERD_use_periodic_ghosts   (only need be done once for the model)
*    USERD_get_periodic_ghosts_num_symmetry_faces
*
*   Simple Structured Examples:
*
*   periodic translation in i  (min i goes to max i)
*   ================================================
*
*               j          (in terms of original elements)
*
*     *- - - - -*---------*---------*---------*- - - - -*
*     .  ghost  |         |         |         |  ghost  .
*     .    8    |    6    |    7    |    8    |    6    .
*     .         |         |         |         |         .
*     *- - - - -*---------*---------*---------*- - - - -*
*     .  ghost  |         |         |         |  ghost  .
* near.    5    |    3    |    4    |    5    |    3    .  far
*     .         |         |         |         |         .
*     *- - - - -*---------*---------*---------*- - - - -*
*     .  ghost  |         |         |         |  ghost  .
*     .    2    |    0    |    1    |    2    |    0    .
*     .         |         |         |         |         .
*     *- - - - -*---------*---------*---------*- - - - -*   i
*
*              <===== translation direction =====>
*
*     num_symmetry_faces = 2;
*     for face 0:
*        which_ijk = 1
*        min_or_max = 0
*     for face 1:
*        which_ijk = 1
*        min_or_max = 1
*
*
*   periodic rotation of 90 degrees about k  (min j goes to max j)
*   ==============================================================
*
*               15          12            9
*              *-----------*------------*
*              |                      . |
*              |         .          .   |           Not as easy to
*              |                  .     |           show!
```

```
 *               |       .        .       |
 *   far         |                .       |
 *   face     14 |   11.     8 .           | 6
 *  (max j)      *----*-----*             *
 *               |   .    . |          .   |
 *               |      .   | 5  .         |
 *               |  . .     *              |
 *               |  .   .   |              |     ^
 *               |..        | 2            | 3   |
 *               *----------*--------------*     j
 *       1,4,7,10,13
 *               i ->
 *                       near face (min j)
 *
 *            rotates in plane about k
 *
 *     num_faces = 2
 *     for face 0:
 *        which_ijk = 2
 *        min_or_max = 0
 *     for face 1:
 *        which_ijk = 2
 *        min_or_max = 1
 *
 *
 *   mirror in i, and j (so min i reflects, ans min j reflects)
 *   ==========================================================
 *
 *               ^
 *               |             (in terms of original elements
 *               j
 *
 *     *- - - - -*---------*---------*---------*
 *     . ghost   |         |         |         |
 *     .    6    |    6    |    7    |    8    |
 *     .         |         |         |         |
 *     *- - - - -*---------*---------*---------*
 *     .  ghost  |         |         |         |
 *     .    3    |    3    |    4    |    5    |
 *     .         |         |         |         |
 *     *- - - - -*---------*---------*---------*
 *     .  ghost  |         |         |         |
 *     .    0    |    0    |    1    |    2    |
 *     .         |         |         |         |
 *     *- - - - -*---------*---------*---------*  i -->
 *     .  ghost  .  ghost  .  ghost  .  ghost  .
 *     .    0    .    0    .    1    .    2    .
 *     .         .         .         .         .
 *     *- - - - -*- - - - -*- - - - -*- - - - -*
 *
 *     num_faces = 2
 *     for face 0:
 *        which_ijk = 1;
 *        min_or_max = 0;
 *     for face 1:
 *        which_ijk = 2;
 *        min_or_max = 0;
 *-----------------------------------------------------------------------*/
int
USERD_get_periodic_ghosts_structured_face_info(int part_number,
                                               int face,
```

```
int *providing_matrix,
double m[4][4],
int *which_ijk,
int *min_or_max)
```

```
/*----------------------------------------------------------------------
 USERD_get_reader_descrip
 *                              <optional>   (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the description of the reader, so gui can give more info
 *
 *  (OUT) reader_descrip       = the description of the reader
 *                              (max length is MAXFILENP, which
 *                               is 255)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *----------------------------------------------------------------------*/
int
USERD_get_reader_descrip(char descrip[Z_MAXFILENP])
```

```
/*----------------------------------------------------------------------
 USERD_get_reader_release
 *                                <optional>  (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the release string for the reader.
 *
 *   This release string is a free-format string which is for
 *   informational purposes only.  It is often useful to increment
 *   the release number/letter to indicate a change in the reader.
 *   The given string will simply be output by the EnSight server
 *   when the reader is selected.
 *
 *  (OUT) release_number       = the release number of the reader
 *                             (max length is Z_MAX_USERD_NAME, which
 *                              is 20)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Called when the reader is selected for use.
 *----------------------------------------------------------------------*/
int
USERD_get_reader_release(char version_number[Z_MAX_USERD_NAME])
```

```
/*----------------------------------------------------------------------
 USERD_get_reader_version
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Gets the API version number of the user defined reader
 *
 *   The functions that EnSight will call depends on this API
 *   version.  See the README files for more information.
 *
 *  (OUT) version_number       = the version number of the reader
 *                             (max length is Z_MAX_USERD_NAME, which
 *                              is 20)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Always called.
 *
 *  * This needs to be "2.000" or greater. Otherwise EnSight will assume
 *    this reader is API 1.0 instead of 2.0
 *----------------------------------------------------------------------*/
int
USERD_get_reader_version(char version_number[Z_MAX_USERD_NAME])
```

```
/*----------------------------------------------------------------------
 USERD_get_sol_times
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Get the solution times associated with each time step for desired timeset.
 *
 *  (IN)  timeset_number     = the timeset number  (1 based)
 *
 *                             For example: If USERD_get_number_of_timesets
 *                                          returns 2, the valid
 *                                          timeset_number's would be 1 and 2.
 *
 *  (OUT) solution_times     = 1D array of solution times per time step
 *
 *                                (Array will have been allocated
 *                                 Num_time_steps[timeset_number] long)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * These must be non-negative and increasing.
 *----------------------------------------------------------------------*/
int
USERD_get_sol_times(int timeset_number,
                    float *solution_times)
```

```
/*----------------------------------------------------------------------
 USERD_get_structured_reader_cinching
 *                                              (Version 2.06 and later)
 *----------------------------------------------------------------------
 *
 *  Gets whether this reader will do structured cinching for block data
 *  This means that it will handle the min, max, and step values for a
 *  given block and return the coordinate components or variable components
 *  in their "cinched" state when partial extraction or striding is used.
 *  This is as opposed to returning the entire component (ignoring min, max
 *  and stride) and letting Ensight pick out the values actually used.
 *
 *  returns: Z_OK    if the reader will handle the
 *                   min, max, and stride and return
 *                   the cinched values only.
 *
 *           Z_UNDEF or Z_ERR if will return entire component
 *                            and rely on EnSight to cinch.
 *
 * Notes:
 *  Unless you can actually pull out the desired min, max, and stride
 *  without using a full component of memory, don't enable this feature.
 *
 * IMPORTANT!!  If your reader will be used for structured auto-distribute,
 *              you must implement this feature, which includes this routine and
 *              USERD_set_block_range_and_stride
 *----------------------------------------------------------------------*/
int
USERD_get_structured_reader_cinching( void )
```

```
/*----------------------------------------------------------------------
  USERD_get_timeset_description
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  Get the description to associate with the desired timeset.
 *
 *  (IN)  timeset_number      = the timeset number
 *
 *                              For example: If USERD_get_number_of_timesets
 *                                           returns 2, the valid
 *                                           timeset_number's would be 1 and 2.
 *
 *  (OUT) timeset_description  = timeset description string
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * A string of NULLs is valid for timeset_description
 *----------------------------------------------------------------------*/
int
USERD_get_timeset_description(int timeset_number,
                              char timeset_description[Z_BUFL])
```

```
/*----------------------------------------------------------------------
 USERD_get_uns_failed_params
 *                                          (Version 2.04 and later)
 *----------------------------------------------------------------------
 *
 * Provides the failure variable and and failure criteria for failed elements
 *
 * (OUT) fail_var_name         = Variable name to be used for failure.
 *                               Must be a per-elem scalar!
 *
 * (OUT) threshold_val1        = 1st number for failure comparison
 *                                Always used in the determination.
 *                                If threshold_operator1 is Z_ELE_FAILED_EQUAL,
 *                                then only this threshold value is used.
 *
 * (OUT) threshold_val2        = 2nd number for failure comparison
 *                                 Will be used if threshold_operator1 is not
 *                                 set to Z_ELE_FAILED_EQUAL and
 *                                 logic_criteria2 is set to
 *                                    Z_ELE_FAILED_LOGIC_AND or
 *                                    Z_ELE_FAILED_LOGIC_OR
 *
 * (OUT) threshold_operator1   = 1st threshold operator
 *                                    Z_ELE_FAILED_GREATER        - greater than
 *                                    Z_ELE_FAILED_LESS           - less than
 *                                    Z_ELE_FAILED_EQUAL          - equal
 *                                    Z_ELE_FAILED_NOT_EQUAL      - not equal
 *                                 Sets the logic for use of threshold_val1
 *
 * (OUT) threshold_operator2   = 2nd threshold operator
 *                                    Z_ELE_FAILED_GREATER        - greater than
 *                                    Z_ELE_FAILED_LESS           - less than
 *                                    Z_ELE_FAILED_EQUAL          - equal
 *                                    Z_ELE_FAILED_NOT_EQUAL      - not equal
 *                                 Used if logic_criteria2 is set to
 *                                   Z_ELE_FAILED_LOGIC_AND or
 *                                   Z_ELE_FAILED_LOGIC_OR
 *                                 Sets the loginc for use of threshold_val2
 *
 * (OUT) logic_criteria2       = Determines if using second criteria and if it
 *                               is an and or or condition
 *                                    Z_ELE_FAILED_LOGIC_NONE
 *                                    Z_ELE_FAILED_LOGIC_AND
 *                                    Z_ELE_FAILED_LOGIC_OR
 *
 * returns: TRUE  if failed elements should be used
 *          FALSE if not using failed elements
 *
 *
 * Notes:
 *
 *   Example 1:
 *       If variable "failure" is an element scalar that has values which
 *       are either 0.0 (for not-failed) or 1.0 (for failed), then:
 *          fail_var_name       = "failure"
 *          threshold_val1      = 1.0
 *          threshold_operator1 = Z_ELE_FAILED_EQUAL
 *             **rest is ignored**
 *
 *
 *
```

```
 *   Example 2:
 *       If variable "Stress" is an element scalar, and failure occurs
 *       when the stress exceeds 3000.0
 *          fail_var_name       = "Stress"
 *          threshold_val1      = 3000.0
 *          threshold_operator1 = Z_ELE_FAILED_GREATER
 *          logic_criteria2     = Z_ELE_FAILED_LOGIC_NONE
 *
 *   Example 3:
 *       If variable "Stress" is an element scalar, and failure occurs
 *       when the value is less than -500, or greater than 400
 *          fail_var_name       = "Stress"
 *          threshold_val1      = -500.0
 *          threshold_operator1 = Z_ELE_FAILED_LESS
 *          threshold_val2      = 400.0
 *          threshold_operator2 = Z_ELE_FAILED_GREATER
 *          logic_criteria2     = Z_ELE_FAILED_LOGIC_OR
 *-------------------------------------------------------------------*/
int USERD_get_uns_failed_params(char *fail_var_name,
                                float *threshold_val1,
                                float *threshold_val2,
                                int *threshold_operator1,
                                int *threshold_operator2,
                                int *logic_criteria2)
```

```
/*----------------------------------------------------------------------
 USERD_get_var_by_component
 *                                          (Version 2.00 and later)
 *                                (Modified at 2.01 as described below)
 *----------------------------------------------------------------------
 *
 *  Gets the values of a variable component.  Both unstructured and structured
 *  parts use this routine.
 *
 *  if Z_PER_NODE:
 *    Get the component value at each node for a given variable in the part.
 *
 *  or if Z_PER_ELEM:
 *    Get the component value at each element of a specific part and type for
 *    a given variable.
 *
 *  (IN)  which_variable          = The variable number (1 to Num_variables)
 *
 *  (IN)  which_part                Since EnSight Version 7.4
 *                                  -------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label
 *                                                loaded in
 *                                               USERD_get_gold_part_build_info.
 *                                               It is NOT the part table index.
 *
 *  (IN)  var_type                = Z_SCALAR
 *                                  Z_VECTOR
 *                                  Z_TENSOR     ( symmetric tensor)
 *                                  Z_TENSOR9    (asymmetric tensor)
 *
 *  (IN)  which_type
 *
 *          if Z_PER_NODE:          Not used
 *
 *          if Z_PER_ELEM:        = The element type
 *                                  Z_POINT    node point element
 *                                  Z_BAR02    2 node bar
 *                                  Z_BAR03    3 node bar
 *                                  Z_TRI03    3 node triangle
 *                                  Z_TRI06    6 node triangle
 *                                  Z_QUA04    4 node quad
 *                                  Z_QUA08    8 node quad
 *                                  Z_TET04    4 node tetrahedron
 *                                  Z_TET10   10 node tetrahedron
 *                                  Z_PYR05    5 node pyramid
 *                                  Z_PYR13   13 node pyramid
 *                                  Z_PEN06    6 node pentahedron
 *                                  Z_PEN15   15 node pentahedron
 *                                  Z_HEX08    8 node hexahedron
 *                                  Z_HEX20   20 node hexahedron
```

```
*
*   Starting at API 2.01:
*   =====================
*                                  Z_G_POINT    ghost node point element
*                                  Z_G_BAR02    2 node ghost bar
*                                  Z_G_BAR03    3 node ghost bar
*                                  Z_G_TRI03    3 node ghost triangle
*                                  Z_G_TRI06    6 node ghost triangle
*                                  Z_G_QUA04    4 node ghost quad
*                                  Z_G_QUA08    8 node ghost quad
*                                  Z_G_TET04    4 node ghost tetrahedron
*                                  Z_G_TET10   10 node ghost tetrahedron
*                                  Z_G_PYR05    5 node ghost pyramid
*                                  Z_G_PYR13   13 node ghost pyramid
*                                  Z_G_PEN06    6 node ghost pentahedron
*                                  Z_G_PEN15   15 node ghost pentahedron
*                                  Z_G_HEX08    8 node ghost hexahedron
*                                  Z_G_HEX20   20 node ghost hexahedron
*
*  (IN)  imag_data                = TRUE if imag component
*                                   FALSE if real component
*
*  (IN)  component                = The component: (0       if Z_SCALAR)
*                                                  (0 - 2   if Z_VECTOR)
*                                                  (0 - 5   if Z_TENSOR)
*                                                  (0 - 8   if Z_TENSOR9)
*
*                                 * 6 Symmetric Indicies, 0:5    *
*                                 * ---------------------------- *
*                                 *     | 11 12 13 |   | 0 3 4 | *
*                                 *     |          |   |       | *
*                                 * T = |    22 23 | = |   1 5 | *
*                                 *     |          |   |       | *
*                                 *     |       33 |   |     2 | *
*
*                                 * 9 General   Indicies, 0:8    *
*                                 * ---------------------------- *
*                                 *     | 11 12 13 |   | 0 1 2 | *
*                                 *     |          |   |       | *
*                                 * T = | 21 22 23 | = | 3 4 5 | *
*                                 *     |          |   |       | *
*                                 *     | 31 32 33 |   | 6 7 8 | *
*
* (OUT) var_array
*
*     ------------------------------------------------------------------------
*     (IMPORTANT: this array is 1-based for both Z_PER_NODE and Z_PER_ELEM!!!
*     ------------------------------------------------------------------------
*
*          if Z_PER_NODE:       = 1D array containing variable component value
*                                 for each node.
*
*                                (Array will have been allocated
*                                 (number_of_nodes+1) long)
*
*              Info stored in this fashion:
*                   var_array[0] = not used
*                   var_array[1] = var component for node 1 of part
*                   var_array[2] = var component for node 2 of part
*                   var_array[3] = var component for node 3 of part
*                   etc.
```

```
 *
 *           if Z_PER_ELEM:      = 1d array containing variable component value
 *                                 for each element of particular part & type.
 *
 *                              (Array will have been allocated
 *                               (number_of_elements[which_part][which_type] + 1)
 *                                long.  See USERD_get_gold_part_build_info)
 *
 *              Info stored in this fashion:
 *                   var_array[1] = var component for elem 1 (of part and type)
 *                   var_array[2] = var component for elem 2         "
 *                   var_array[3] = var conponent for elem 3         "
 *                   etc.
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *       or: Z_UNDEF if this variable is not defined on this part. In which
 *                   case you need not load anything into the var_array.
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Not called unless Num_variables is > 0
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *
 *  * If the variable is not defined for this part, simply return with a
 *    value of Z_UNDEF.  EnSight will treat the variable as undefined for
 *    this part.
 *-----------------------------------------------------------------------*/
int
USERD_get_var_by_component(int which_variable,
                           int which_part,
                           int var_type,
                           int which_type,
                           int imag_data,
                           int component,
                           float *var_array)
```

```
/*----------------------------------------------------------------------
 USERD_get_var_by_component_in_buffers
 *                                <optional>  (Version 2.08 and later)
 *----------------------------------------------------------------------
 *
 *  if Z_PER_NODE:
 *    Get the component value at each node for a given variable in the part
 *    in buffers.
 *
 *  or if Z_PER_ELEM:
 *    Get the component value at each element of a specific part and type for
 *    a given variable in buffers.
 *
 *   This is one of several optional routines than can be added into any
 *   API 2.* reader to be used by the Unstructured Auto Distribute
 *   capability in EnSight 8.2 and later.
 *
 *   Unstructured Auto Distribute is a capability requiring Server of Servers
 *   (SOS) that will partition an unstructured model for you automatically
 *   across a set of servers.
 *
 *   If you do not implement this routine (and the other in_buffers routines)
 *   in your reader, EnSight can still perform this operation but will require
 *   much more memory on each server to read in the data (somewhat like each
 *   server having to read the whole model). You will however, get the execution
 *   advantage of having your model partitioned across multiple servers.
 *
 *   If you do implement this routine (and the other in_buffers routines) in
 *   your reader (in a proper manner), you should be able to not only get the
 *   execution advantages, but also memory usage on each server which is
 *   proportional to the subset that it is assigned to deal with.
 *
 *   Note that this optional routine is functionally quite similar
 *   to the USERD_get_var_by_component routine. And thus its implementation
 *   should not be too difficult to add to any existing reader that has already
 *   implemented the USERD_get_var_by_component routine.
 *
 *  (IN)  which_variable          = The variable number (1 to Num_variables)
 *
 *  (IN)  which_part                Since EnSight Version 7.4
 *                                  -------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label
 *                                                 loaded in
 *                                              USERD_get_gold_part_build_inf\o.
 *                                              It is NOT the part table index.
 *
 *  (IN)  var_type                = Z_SCALAR
 *                                  Z_VECTOR
 *                                  Z_TENSOR     ( symmetric tensor)
 *                                  Z_TENSOR9    (asymmetric tensor)
```

```
*
*  (IN)  which_type
*
*           if Z_PER_NODE:         Not used
*
*           if Z_PER_ELEM:       = The element type
*                                  Z_POINT    node point element
*                                  Z_BAR02     2 node bar
*                                  Z_BAR03     3 node bar
*                                  Z_TRI03     3 node triangle
*                                  Z_TRI06     6 node triangle
*                                  Z_QUA04     4 node quad
*                                  Z_QUA08     8 node quad
*                                  Z_TET04     4 node tetrahedron
*                                  Z_TET10    10 node tetrahedron
*                                  Z_PYR05     5 node pyramid
*                                  Z_PYR13    13 node pyramid
*                                  Z_PEN06     6 node pentahedron
*                                  Z_PEN15    15 node pentahedron
*                                  Z_HEX08     8 node hexahedron
*                                  Z_HEX20    20 node hexahedron
*
*                                  Z_G_POINT    ghost node point element
*                                  Z_G_BAR02    2 node ghost bar
*                                  Z_G_BAR03    3 node ghost bar
*                                  Z_G_TRI03    3 node ghost triangle
*                                  Z_G_TRI06    6 node ghost triangle
*                                  Z_G_QUA04    4 node ghost quad
*                                  Z_G_QUA08    8 node ghost quad
*                                  Z_G_TET04    4 node ghost tetrahedron
*                                  Z_G_TET10   10 node ghost tetrahedron
*                                  Z_G_PYR05    5 node ghost pyramid
*                                  Z_G_PYR13   13 node ghost pyramid
*                                  Z_G_PEN06    6 node ghost pentahedron
*                                  Z_G_PEN15   15 node ghost pentahedron
*                                  Z_G_HEX08    8 node ghost hexahedron
*                                  Z_G_HEX20   20 node ghost hexahedron
*
*                                  Z_NSIDED     n node nsided polygon
*                                  Z_NFACED     n face nfaced polyhedron
*                                  Z_G_NSIDED   n node ghost nsided polygon
*                                  Z_G_NFACED   n face ghost nfaced polyhedron
*
*
*
*  (IN)  imag_data              = TRUE if imag component
*                                 FALSE if real component
*
*
*  (IN)  component              = The component: (0       if Z_SCALAR)
*                                                (0 - 2   if Z_VECTOR)
*                                                (0 - 5   if Z_TENSOR)
*                                                (0 - 8   if Z_TENSOR9)
*
*
*
*                                * 6 Symmetric Indicies, 0:5    *
*                                * ---------------------------- *
*                                *     | 11 12 13 |   | 0 3 4 | *
*                                *     |          |   |       | *
*                                * T = |    22 23 | = |   1 5 | *
```

```
*                                 *     |          |   |       | *
*                                 *     |       33 |   |     2 | *
*
*                                 * 9 General   Indicies, 0:8    *
*                                 * -------------------------- *
*                                 *     | 11 12 13 |   | 0 1 2 | *
*                                 *     |          |   |       | *
*                                 * T = | 21 22 23 | = | 3 4 5 | *
*                                 *     |          |   |       | *
*                                 *     | 31 32 33 |   | 6 7 8 | *
*
*  (IN) ne_beg
*         if Z_PER_NODE:  = Zero based, 1st node index of the buffered set
*         if Z_PER_ELEM:  = Zero based, 1st element index of the buffered set
*
*  (IN) ne_end
*         if Z_PER_NODE:  = Zero based, last node index of the buffered set
*         if Z_PER_ELEM:  = Zero based, last element index of the buffered set
*
*                     Thus, for first five elements or nodes:
*                       e_beg = 0
*                       e_end = 4
*                       total_number = (e_end - e_beg) + 1 = (4 - 0) + 1 = 5
*
*                     for second five elements or nodes, would be:
*                       e_beg = 5
*                       e_end = 9
*                       total_number = (e_end - e_beg) + 1 = (9 - 5) + 1 = 5
*
*                     for all elements or nodes of a part, would be:
*                       n_beg = 0
*                       n_end = num_elements_or_nodes - 1
*
* (IN) first                      = TRUE if first invocation of a buffered set.
*                                   Will be FALSE for all subsequent invocations
*                                   of the set.  This is so you can open files,
*                                   get to the correct starting spot,
*                                   initialize, etc.
*
* (IN) buffer_size                = The size of the buffer.
*                                   Namely:  var_array[buffer_size]
*
* (IN) leftside                   = TRUE if current time is at a timestep or
*                                         when getting the left side of a time
*                                         span that encloses the current time.
*                                 = FALSE when getting the right side of a time
*                                         span that encloses the current time.
*
*  Timeline:
*   step1         step2          step3
*     |-------------|--------------|-------...  requires no interpolation
*                   ^                          get values at step2
*                   current time                  (leftside = TRUE)
*
*
*
*  Timeline:
*   step1         step2          step3
*     |-------------|--------------|-------...  requires interpolation
*               ^                            get values at step1 (leftside = TRUE)
*               current time           and get values at step2 (leftside = FALSE)
```

```
*
*
*  Note that it would generally be easier for this routine if EnSight got all
*       of the left side, then all of the right side, and then did its
*       interpolation. But, in the spirit of doing things in buffers (to save
*       memory) it gets a left side buffer (and the corresponding right side
*       buffer and interpolates these), if needed, before going to the next
*       buffer of the set.  Thus, you need to be able to handle that
*       situation.
*
*  Note also that EnSight will have called the routine to change the current
*       time step between the two invocations when interpolation is required.
*       And Ensight does the interpolating.  This variable is provided so
*       that you can deal with two different files or pointers between the
*       corresponding invocations for the two times
*
* (OUT) var_array
*
*     ------------------------------------------------------------------------
*     (IMPORTANT: this array is 0-based for both Z_PER_NODE and Z_PER_ELEM!!!
*     ------------------------------------------------------------------------
*
*          if Z_PER_NODE:       = 1D buffer array set up to hold a variable
*                                 component value for nodes.
*
*          if Z_PER_ELEM:       = 1D buffer array set up to hold a variable
*                                 component value for elements.
*
*                                 (Array will have been allocated
*                                  buffer_size long)
*
*              Info stored in this fashion:
*                   var_array[0] = var component for node or element 1 of part
*                   var_array[1] = var component for node or element 2 of part
*                   var_array[2] = var component for node or element 3 of part
*                   etc.
*
*
*   * Example, (if 158 quad elements with a real Z_PER_ELEM scalar,
*                current time is between steps, and buffer size is 75)
*
*        first invocation:            (for left side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
*          component = 0
*          ne_beg = 0
*          ne_end = 157
*          first = TRUE               Will be TRUE the first time!
*          buffer_size = 75
*          leftside = TRUE            <==
*          var_array[75]              load in scalar value for elements 1 - 75
*          *num_returned = 75         set this
*          return(0)                  return this (indicates more to do)
*
*
*        second invocation:           (for right side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
*          component = 0
```

```
*          ne_beg = 0
*          ne_end = 157
*          first = TRUE                Note: Will still be TRUE (because is
*                                                              right side)
*          buffer_size = 75
*          leftside = FALSE            <==
*          var_array[75]               load in scalar value for elements 1 - 75
*          *num_returned = 75          set this
*          return(0)                   return this (indicates more to do)
*
* ------------------------------
*        third invocation:             (for left side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
*          component = 0
*          ne_beg = 0
*          ne_end = 157
*          first = FALSE               Will be FALSE now
*          buffer_size = 75
*          leftside = TRUE             <==
*          var_array[75]               load in scalar value for
*                                              elements 76 - 150
*          *num_returned = 75          set this
*          return(0)                   return this (indicates more to do)
*
*        fourth invocation:            (for right side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
*          component = 0
*          ne_beg = 0
*          ne_end = 157
*          first = FALSE
*          buffer_size = 75
*          leftside = FALSE            <==
*          var_array[75]               load in scalar value for
*                                              elements 76 - 150
*          *num_returned = 75          set this
*          return(0)                   return this (indicates more to do)
*
*-----------------------------------
*        fifth invocation:             (for left side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
*          component = 0
*          ne_beg = 0
*          ne_end = 157
*          first = FALSE               Will still be FALSE
*          buffer_size = 75
*          leftside = TRUE             <==
*          var_array[75]               load in scalar value
*                                              for elements 151 - 158
*          *num_returned = 8           set this
*          return(1)                   return this (indicates no more to do)
*
*        sixth invocation:            (for right side of time span)
*          var_type = Z_SCALAR
*          which_type = Z_PER_ELEM
*          imag_data = FALSE
```

```
 *          component = 0
 *          ne_beg = 0
 *          ne_end = 157
 *          first = FALSE
 *          buffer_size = 75
 *          leftside = FALSE           <==
 *          var_array[75]              load in scalar value
 *                                             for elements 151 - 158
 *          *num_returned = 8          set this
 *          return(1)                  return this (indicates no more to do)
 *
 *
 *  (OUT)  *num_returned       = The number of nodes or elements whose variable values
 *                               are returned in the buffer. This will normally be
 *                               equal to buffer_size except for that last buffer -
 *                               which could be less than a full buffer
 *
 * returns  0  if got some, more to do
 *          1  if got some, done
 *         -1  if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *
 *  * Again, make sure each buffer is zero based.
 *    For our example using buffers above:
 *
 *                                      Invocation:
 *               ----------------   ------------------   -------------------
 *                  1        2         3          4          5          6
 *               -------  -------   --------  --------   ---------  ---------
 * var_array[0]  quad 1L  quad 1R   quad 76L  quad 76R   quad 151L  quad 151R
 *
 * var_array[1]  quad 2L  quad 2R   quad 77L  quad 77R   quad 152L  quad 152R
 *
 * ...
 *
 * var_array[74] quad 75L quad 75R  quad 150L quad 150R  quad 158L  quad 158R
 *
 *   Where:   L indicates left time step
 *            R indicates right time step
 *-------------------------------------------------------------------*/
int
USERD_get_var_by_component_in_buffers(int which_variable,
                                      int which_part,
                                      int var_type,
                                      int which_type,
                                      int imag_data,
                                      int component,
                                      float *var_array,
                                      int first,
                                      int ne_beg,
                                      int ne_end,
                                      int buffer_size,
                                      int leftside,
                                      int *num_returned)
```

```
/*-----------------------------------------------------------------------
 USERD_get_var_extract_gui_defaults
 *                                        <optional> (version 2.05 and later)
 *-----------------------------------------------------------------------
 *
 *  This routine defines the Titles, status, List choices, strings, etc that
 *  are fed up to the GUI for that after read situation.  (It is very similar
 *  to the USERD_get_extra_gui_defaults routine, which occurs before the read)
 *
 *   (OUT) toggle_Title               = title for each toggle
 *                                      array dimension is
 *                                  [num_toggles] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) toggle_default_status      = Setting for each toggle (TRUE or FALSE)
 *                                      array dimension is [num_toggles] long
 *
 *   (OUT) pulldown_Title             = title for each pulldown
 *                                      array dimension is
 *                                  [num_pulldowns] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) pulldown_number_in_list    = number of items in each pulldown
 *                                      array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_default_selection = item selection for each pulldown
 *                                      array dimension is [num_pulldowns] long
 *
 *   (OUT) pulldown_item_strings      = pulldown item strings
 *                                      array is [num_pulldowns] by
 *                                             [Z_MAX_NUM_GUI_PULL_ITEMS] by
 *                                             [Z_LEN_GUI_PULL_STR] long
 *
 *   (OUT) field_Title                = title for each field
 *                                      array dimension is
 *                                  [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *   (OUT) field_user_string          = content of the field
 *                                      array dimension is
 *                                  [num_fields] by [Z_LEN_GUI_TITLE_STR] long
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * The library is loaded, this routine is called,
 *    then the library is unloaded.
 *
 *  * Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 * ---------------------------------------------- */
int USERD_get_var_extract_gui_defaults(char **toggle_Title,
                                       int *toggle_default_status,
                                       char **pulldown_Title,
                                       int *pulldown_number_in_list,
                                       int *pulldown_default_selection,
                                       char ***pulldown_item_strings,
                                       char **field_Title,
                                       char **field_user_string)
```

```
/*-----------------------------------------------------------------------
 USERD_get_var_extract_gui_numbers
 *                                  <optional> (version 2.05 and later)
 * ----------------------------------------------------------------------
 *   The Var_Extract GUI routines are added to allow the user to customize a
 *   extraction parameters for variable "after" the file has been read.
 *   These things can be modified and the variables will be updated/refreshed
 *   according to the new parameters.
 *   (It is similar to the USERD_get_extra_gui_numbers routine)
 *
 *   This routine defines the numbers of toggles, pulldowns & fields
 *
 *   (OUT) num_Toggles    = number of toggles that will be provided
 *
 *   (OUT) num_pulldowns  = number of pulldowns that will be provided
 *
 *   (OUT) num_fields     = number of fields that will be provided
 *
 *   Notes:
 *   * There are three routines that work together:
 *         USERD_get_var_extract_gui_numbers
 *         USERD_get_var_extract_gui_defaults
 *         USERD_set_var_extract_gui_data
 *
 *     The existence of these routine indicates that
 *     you wish to have the Var Extract capability.
 *
 *     If you don't want the Var Extract GUI features,
 *     simply delete these routines, or change their
 *     names to something such as
 *     USERD_DISABLED_get_var_extract_gui_defaults
 *
 *     The presence of these routines
 *     will ensure that EnSight will call them and
 *     use their data to modify the extraction parameters
 *     with some or all of the following:
 *     toggles, pulldown menu and fields.
 *
 *     The user can then interact with the var extract portion of the
 *     GUI and then send their choices to
 *     USERD_set_var_extract_gui_data
 *
 *     Therefore if USERD_get_var_extract_gui_numbers
 *     exists then the other two must exist.
 *
 *     If none exist, then the GUI will be unchanged.
 *
 *     Toggle data will return an integer
 *                                TRUE if checked
 *                                FALSE if unchecked
 *
 *     Pulldown menu will return an integer representing
 *                                the menu item selected
 *
 *     Field will return a string Z_LEN_GUI_FIELD_STR long.
 *
 * * The following are defined in the global_extern.h
 *          Z_MAX_NUM_GUI_PULL_ITEMS max num GUI pulldowns
 *          Z_LEN_GUI_PULL_STR  max length of GUI pulldown string
 *          Z_LEN_GUI_FIELD_STR  max length of field string
 *          Z_LEN_GUI_TITLE_STR   max length of title string
```

```
 *
 * * The library is loaded, this routine is called,
 *   then the library is unloaded.
 *
 * *  Do not define globals in this routine as when the library is unloaded,
 *    you'll lose them.
 *-------------------------------------------------------------------------------*/
void USERD_get_var_extract_gui_numbers(int *num_Toggles,
                                       int *num_pulldowns,
                                       int *num_fields)
```

```
/*----------------------------------------------------------------------
 USERD_get_var_value_at_specific
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  if Z_PER_NODE:
 *    Get the value of a particular variable at a particular node in a
 *    particular part at a particular time.
 *
 *  or if Z_PER_ELEM:
 *    Get the value of a particular variable at a particular element of
 *    a particular type in a particular part at a particular time.
 *
 *  (IN)  which_var   = Which variable (1 to Num_variables)
 *
 *  (IN)  which_node_or_elem
 *
 *                    If Z_PER_NODE:
 *                      = The node number.  This is not the id, but is
 *                                          the index of the node
 *                                          list (1 based), or the block's
 *                                          node list (1 based).
 *
 *                          Thus,  coord_array[1]
 *                                 coord_array[2]
 *                                 coord_array[3]
 *                                      .      |
 *                                      .      |which_node_or_elem index
 *                                      .             ----
 *
 *
 *                    If Z_PER_ELEM:
 *                      = The element number. This is not the id, but is
 *                                            the element number index
 *                                            of the number_of_element array
 *                                       (see USERD_get_gold_part_build_info),
 *                                            or the block's element list
 *                                            (1 based).
 *
 *                          Thus,  for which_part:
 *                                 conn_array[which_elem_type][0]
 *                                 conn_array[which_elem_type][1]
 *                                 conn_array[which_elem_type][2]
 *                                      .                       |
 *                                      .          (which_node_or_elem - 1) index
 *                                      .                           ----
 *
 *
 *
 *  (IN)  which_part                Since EnSight Version 7.4
 *                                  -------------------------
 *                                = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that
 *                                   is loaded in USERD_get_gold_part_build_info)
 *
 *
```

```
 *                                  Prior to EnSight Version 7.4
 *                                  ----------------------------
 *                                = The part id   This is the part_id label
 *                                                loaded in
 *                                             USERD_get_gold_part_build_info.
 *                                             It is NOT the part table index.
 *
 *  (IN)  which_elem_type
 *
 *                        If Z_PER_NODE, or block part:
 *                          = Not used
 *
 *                        If Z_PER_ELEM:
 *                          = The element type.  This is the element type index
 *                                               of the number_of_element array
 *                                          (see USERD_get_gold_part_build_info)
 *
 *  (IN)  time_step   = Time step to use  (0 to Num_time_steps[the proper var timeset])
 *
 *  (IN)  imag_data   = TRUE if want imaginary data file.
 *                      FALSE if want real data file.
 *
 *  (OUT) values      = scalar or vector component value(s)
 *                        values[0] = scalar or vector[0]
 *                        values[1] = vector[1]
 *                        values[2] = vector[2]
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *           Z_NOT_IMPLEMENTED if not implemented and want to use the slower,
 *                             complete update method within EnSight.
 *
 *  Notes:
 *  * This routine is used in node querys over time (or element querys over
 *    time for Z_PER_ELEM variables).  If these operations are not critical
 *    to you, this can be a dummy routine.
 *
 *  * The per_node or per_elem classification must be obtainable from the
 *    variable number (a var_classify array needs to be retained)
 *
 *  * The time step given is for the proper variable timeset. Thus, it
 *    must be obtainable from the variable number also.
 *-----------------------------------------------------------------------*/
int
USERD_get_var_value_at_specific(int which_var,
                                int which_node_or_elem,
                                int which_part,
                                int which_elem_type,
                                int time_step,
                                float values[3],
                                int imag_data)
```

```
/*----------------------------------------------------------------------------
 USERD_get_vglyph_counts
 *                                        <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the counts for the number of vector glyphs in the model
 *
 *  (OUT) num_vglyph_vectors   = The number of vector glyphs in the model.
 *                               (Must be set to zero if none)
 *
 *  (OUT) num_vglyph_timelines = The number of vector glyph timelines in the model.
 *                               (can be zero if all vector glyphs are static)
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *--------------------------------------------------------------------------*/
int
USERD_get_vglyph_counts(int *num_vglyph_vectors,
                        int *num_vglyph_timelines)
```

```
/*----------------------------------------------------------------------------
 USERD_get_vglyph_timeline_info
 *                                          <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the vector glyph timeline metadata.
 *
 *  (IN)  vtl        = The vector glyph timeline index
 *                       0 ... (Num_VGLYPH_Timelines-1)
 *
 *  (OUT) id         = The vector glyph timeline id listed in the file
 *
 *  (OUT) numtimes   = The number of times in the vector glyph timeline
 *
 *  (OUT) before     = If EnSight time is before the first step in the
 *                     vector glyph timeline, a value of
 *                        VG_UNDEF  makes the glyph undefined
 *                        VG_NEAREST makes it act as if it is at the first step
 *
 *  (OUT) amidst     = If EnSight time is between steps in the vector glyph timeline,
 *                        VG_INTERPOLATE interpolates values linearly between them
 *                        VG_NEAREST snaps to the closest step for its values.
 *
 *  (OUT) after      = If EnSight time is after the last step in the
 *                     vector glyph timeline, a value of
 *                        VG_UNDEF  makes the glyph undefined
 *                        VG_NEAREST makes it act as if it is at the last step
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *--------------------------------------------------------------------------*/
int
USERD_get_vglyph_timeline_info(int vtl,
                               int *id,
                               int *numtimes,
                               int *before,
                               int *amidst,
                               int *after)
```

```
/*----------------------------------------------------------------------------
  USERD_get_vglyph_timeline_times
 *                                          <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the vector glyph timeline times
 *
 *  (IN)  vtl        = The vector glyph timeline index
 *                       0 ... (Num_VGLYPH_Timelines-1)
 *
 *  (OUT) times      = The array of times in the vector glyph timeline
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *--------------------------------------------------------------------------*/
int
USERD_get_vglyph_timeline_times(int vtl,
                                float *times)
```

```
/*----------------------------------------------------------------------------
 USERD_get_vglyph_vector_info
 *                                          <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the vector glyph metadata
 *
 *  (IN)  vg              = The vector glyph index
 *                          0 ... (Num_VGLYPH_Vectors-1)
 *
 *  (OUT) id              = The vector glyph id number listed in the file
 *
 *  (OUT) description     = The vector glyph description.
 *                          *** Note: This is what a user will see in
 *                                    the GUI to select the glyph
 *
 *  (OUT) time_condition  = VG_STATIC  if vector glyph is always present,
 *                                     no timeline assiciated with it.
 *                          VG_TRANSIENT if vector glyph is associated with
 *                                       a timeline
 *
 *  (OUT) time_line       = The external timeline index of the vector glyph,
 *                            used on if time_condition is VG_TRANSIENT
 *
 *  (OUT) part            = The part that the vector glyph is associated with
 *
 *  (OUT) nidloc          = The node id number where the vector glyph is located.
 *                            (must be -1 if not located at a node)
 *
 *  (OUT) eidloc          = The element id number where the vector glyph is located.
 *                            (must be -1 if not located at an element)
 *
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *     For vector glyphs locates at an xyz location, make sure both nidloc and eidloc
 *     are both set to -1
 *---------------------------------------------------------------------------*/
int
USERD_get_vglyph_vector_info(int vg,
                             int *id,
                             char *description,
                             int *type,
                             int *time_condition,
                             int *time_line,
                             int *part,
                             int *nidloc,
                             int *eidloc)
```

```
/*----------------------------------------------------------------------------
 USERD_get_vglyph_vector_values
 *                                          <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the vector glyph vector component values
 *
 *  (IN)  vg         = The vector glyph index
 *                      0 ... (Num_VGLYPH_Vectors-1)
 *
 *  (OUT) values     = The vector glyph component vector values
 *                     (2D array, with 3 values per timestep)
 *
 *                     for VG_STATIC vector glyph, would be 1 x 3 array
 *                       [xc, yc, zc]
 *
 *                     for VG_TRANSIENT vector glyph, with a timeline
 *                       that has 5 time steps, would be 5 x 3 array
 *                       [xc1, yc1, zc1]
 *                       [xc2, yc2, zc2]
 *                       [xc3, yc3, zc3]
 *                       [xc4, yc4, zc4]
 *                       [xc5, yc5, zc5]
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *--------------------------------------------------------------------------*/
int
USERD_get_vglyph_vector_values(int vg,
                               float **values)
```

```
/*----------------------------------------------------------------------------
 USERD_get_vglyph_vector_xyzloc
 *                                          <optional>  (Version 2.09 and later)
 *----------------------------------------------------------------------------
 *
 *  Gets the vector glyph xyz locations
 *
 *  (IN)  vg         = The vector glyph index
 *                      0 ... (Num_VGLYPH_Vectors-1)
 *
 *  (OUT) xyzloc     = The vector glyph xyz location(s)
 *                     (2D array, with 3 values per timestep)
 *
 *                     for VG_STATIC vector glyph, would be 1 x 3 array
 *                       [x, y, z]
 *
 *                     for VG_TRANSIENT vector glyph, with a timeline
 *                       that has 5 time steps, would be 5 x 3 array
 *                       [x1, y1, z1]
 *                       [x2, y2, z2]
 *                       [x3, y3, z3]
 *                       [x4, y4, z4]
 *                       [x5, y5, z5]
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *--------------------------------------------------------------------------*/
int
USERD_get_vglyph_vector_xyzloc(int vg,
                               float **xyzloc)
```

```
/*----------------------------------------------------------------------------
 USERD_get_xy_query_data
 *                                          <optional>  (Version 2.08 and later)
 *----------------------------------------------------------------------------
 *
 * Gets the xy values for a particular xy_query
 *
 * (IN)  query_num            = query number (zero based)
 *                              (0 to one less than the number of querys
 *                               returned in USERD_get_num_xy_queries)
 *
 * (IN)  num_vals             = number of xy pairs in the query
 *
 * (OUT) xvals                = array of x values, dimensioned to num_vals (0 based)
 *
 * (OUT) yvals                = array of y values, dimensioned to num_vals (0 based)
 *
 * returns: Z_OK  if successful
 *          Z_ERR if a problem
 *
 * Notes:
 *----------------------------------------------------------------------------*/
int USERD_get_xy_query_data(
        int query_num,
        int num_vals,
        float *xvals,
        float *yvals)
```

```
/*----------------------------------------------------------------------------
  USERD_get_xy_query_info
 *                                        <optional>  (Version 2.08 and later)
 *----------------------------------------------------------------------------
 *
 * Gets name, axis titles, and number of xy pairs for a particular xy_query
 *
 * (IN)  query_num            = query number (zero based)
 *                              (0 to one less than the number of querys
 *                               returned in USERD_get_num_xy_queries)
 *
 * (OUT) query_name           = Name for the xy query. (80 chars long)
 *
 * (OUT) query_xtitle         = Title for x axis       (80 chars long)
 *
 * (OUT) query_ytitle         = Title for y axis       (80 chars long)
 *
 * (OUT) query_num_pairs      = number of xy pairs
 *
 * returns: Z_OK  if successful
 *          Z_ERR if a problem
 *
 * Notes:
 *----------------------------------------------------------------------------*/
int USERD_get_xy_query_info(int query_num,
                            char *query_name,
                            char *query_xtitle,
                            char *query_ytitle,
                            int *query_num_pairs )
```

```
/*----------------------------------------------------------------------
 USERD_load_matf_data
 *                                              (Version 2.03 and later)
 *----------------------------------------------------------------------
 *   Get the material id list, mixed-material id list, or
 *   mixed-material values list for the given material set and part (and
 *   element type if material id list)
 *
 *  (IN)  set_index         = the material set index (zero based)
 *
 *  (IN)  part_id           = the part number desired
 *
 *  (IN)  wtyp              = the element type     (used for Z_MAT_INDEX only)
 *
 *                                Z_POINT    node point element
 *                                Z_BAR02    2 node bar
 *                                Z_BAR03    3 node bar
 *                                Z_TRI03    3 node triangle
 *                                Z_TRI06    6 node triangle
 *                                Z_QUA04    4 node quad
 *                                Z_QUA08    8 node quad
 *                                Z_TET04    4 node tetrahedron
 *                                Z_TET10   10 node tetrahedron
 *                                Z_PYR05    5 node pyramid
 *                                Z_PYR13   13 node pyramid
 *                                Z_PEN06    6 node pentahedron
 *                                Z_PEN15   15 node pentahedron
 *                                Z_HEX08    8 node hexahedron
 *                                Z_HEX20   20 node hexahedron
 *                                Z_NSIDED  nsided polygon
 *                                Z_NFACED  nfaced polyhedron
 *
 *                                Z_G_POINT    ghost node point element
 *                                Z_G_BAR02    2 node ghost bar
 *                                Z_G_BAR03    3 node ghost bar
 *                                Z_G_TRI03    3 node ghost triangle
 *                                Z_G_TRI06    6 node ghost triangle
 *                                Z_G_QUA04    4 node ghost quad
 *                                Z_G_QUA08    8 node ghost quad
 *                                Z_G_TET04    4 node ghost tetrahedron
 *                                Z_G_TET10   10 node ghost tetrahedron
 *                                Z_G_PYR05    5 node ghost pyramid
 *                                Z_G_PYR13   13 node ghost pyramid
 *                                Z_G_PEN06    6 node ghost pentahedron
 *                                Z_G_PEN15   15 node ghost pentahedron
 *                                Z_G_HEX08    8 node ghost hexahedron
 *                                Z_G_HEX20   20 node ghost hexahedron
 *                                Z_G_NSIDED  ghost nsided polygon
 *                                Z_G_NFACED  ghost nfaced polyhedron
 *
 *  (IN)  mat_type          = Z_MAT_INDEX for material ids list
 *                            Z_MIX_INDEX for mixed-material ids list
 *                            Z_MIX_VALUE for mixed-material values list
 *                            Z_SPE_VALUE for material species values list
 *
 *  (OUT) ids_list          = If mat_type is Z_MAT_INDEX:
 *                            ---------------------------
 *                             1D material id list
 *                              (Int array will have been allocated
 *                               the appropriate size, as returned in
 *                               USERD_size_matf_data for mat_type Z_MAT_INDEX)
```

```
 *
 *                              If mat_type is Z_MIX_INDEX:
 *                              ---------------------------
 *                               1D mixed-material id list
 *                                 (Int array will have been allocated
 *                                  the appropriate size, as returned in
 *                                  USERD_size_matf_data for mat_type Z_MIX_INDEX)
 *
 *  (OUT) val_list          = 1D mixed-materials values list
 *                              (only used if mat_type is Z_MIX_VALUE)
 *
 *                               (Float array will have been allocated
 *                                the appropriate size, as returned in
 *                                USERD_size_matf_data for mat_type Z_MIX_VALUE)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *  * Will not be called if Num_material_sets is zero,
 *     or Num_materials[set_index] is zero,
 *     or the appropriate size from USERD_size_matf_data is zero
 *  * This function is used only if USERD_get_matf_set_type returns
 *     Z_MISET_VIA_SPARSE_MIX.
 *  * This function is NOT used if USERD_get_matf_set_type returns
 *     Z_MISET_VIA_ESCAL_VARS (and thus is incompatible with
 *     USERD_get_matf_escalars_desc)
 *-------------------------------------------------------------------------*/
int
USERD_load_matf_data( int set_index,
                      int part_id,
                      int wtyp,
                      int mat_type,
                      int *ids_list,
                      float *val_list)
```

```
/*-----------------------------------------------------------------------
 USERD_prefer_auto_distribute
 *                                 <optional> (Version 2.07 and later)
 *-----------------------------------------------------------------------
 *
 * Returns whether the reader will do its own partitioning for SOS
 *
 * returns: TRUE  if prefers to do its own partitioning for SOS
 *          FALSE if EnSight will be asked to do the partitioning
 *                if an auto-distribute is specified
 *
 * Notes:
 *-----------------------------------------------------------------------*/
int
USERD_prefer_auto_distribute(void)
```

```
/*----------------------------------------------------------------------
  USERD_rigidbody_existence
 *                                            (Version 2.05 and later)
 *----------------------------------------------------------------------
 *
 *  Gets the existence of rigid body values or not in the model
 *
 *  returns: Z_OK    if rigid body values exist for the model
 *           Z_UNDEF if no rigid body values exist
 *           Z_ERR   if an error
 *
 *  Notes:
 *  * This will be based on Current_time_step
 *----------------------------------------------------------------------*/
int
USERD_rigidbody_existence( void )
```

```
/*-----------------------------------------------------------------
 USERD_rigidbody_values
 *                                          (Version 2.05 and later)
 *                                (Modified at 2.08 as described below)
 *-----------------------------------------------------------------
 *
 *  Gets the rigid body values for each part
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                   It is NOT the part_id that is
 *                                   loaded in USERD_get_gold_part_build_info)
 *
 *  (OUT) values             values[0] = IX   (x location)
 *                           values[1] = IY   (y location)
 *                           values[2] = IZ   (z location)
 *                           values[3] = E0   (e0 euler value)
 *                           values[4] = E1   (e1 euler value)
 *                           values[5] = E2   (e2 euler value)
 *                           values[6] = E3   (e3 euler value)
 *
 *                           The next 3 are for an optional cg offset. If none
 *                           is needed or desired (namely the first 7 values
 *                           above contain all that is needed), then these
 *                           should be set to 0.0
 *
 *                           values[7]  = xoff   (initial cg x offset)
 *                           values[8]  = yoff   (initial cg y offset)
 *                           values[9]  = z0ff   (initial cg z offset)
 *
 *  Starting at Version 2.08
 *  ========================
 *                       The next 4 values are for and optional initial yaw,
 *                       pitch, roll operation.  This is useful to get parts
 *                       from one standard layout to a different standard
 *                       layout.
 *                       (example, flex body parts computed in an axis system
 *                       different than that of rigid body parts manipulation)
 *                       If not needed or desired, set these to 0.0
 *
 *                       values[10] = rot_order  (order the roations are applied
 *                                                0.0  = no rotations
 *                                                1.0  = xyz order
 *                                                2.0  = xzy order
 *                                                3.0  = yxz order
 *                                                4.0  = yzx order
 *                                                5.0  = zxy order
 *                                                6.0  = zyx order)
 *                       values[11] = xrot   (initial x rotation - degrees)
 *                       values[12] = yrot   (initial y rotation - degrees)
 *                       values[13] = zrot   (initial z rotation - degrees)
 *
 *  returns: Z_OK    if rigid body values sent for this part
 *           Z_UNDEF if no rigid body values exist for this part
 *           Z_ERR   if an error
 *
 *
```

```
 *  Notes:
 *  * This will be based on Current_time_step
 *  * It will not be called unless USERD_rigidbody_existence indicates
 *     that there are some values in the model by returning Z_OK.
 *  * Order that transformations will be applied is:
 *         1. The yaw,pitch,roll rotations, if present
 *               (values[11] through values[13]
 *                in the order specified in rot_order, values[10])
 *         2. The cg offsets, if present     (values[7] through values[9])
 *         3. The euler parameter rotations  (values[3] through values[6])
 *         4. The translations               (values[0] through values[2])
 *
 *-------------------------------------------------------------------------*/
int
USERD_rigidbody_values(int part_number,
                       float values[14])    /* Prior to Version 2.08,
                                               float values[10] */
```

```
/*----------------------------------------------------------------------
 USERD_set_block_range_and_stride
 *                                              (Version 2.06 and later)
 *----------------------------------------------------------------------
 *  Sets the min, max, and step values in each of the i, j, and k, directions
 *  for the given part.
 *
 *  (IN)  part_number             = The part number
 *
 *                                  (1-based index of part table, namely:
 *
 *                                     1 ... Numparts_available.
 *
 *                                  It is NOT the part_id that is
 *                                  loaded in USERD_get_gold_part_build_info)
 *
 *  (IN) mini        = min i plane desired  (zero based)
 *       maxi        = max i plane desired  (zero based)
 *       stepi       = i stride
 *       minj        = min j plane desired  (zero based)
 *       maxj        = max j plane desired  (zero based)
 *       stepj       = j stride
 *       mink        = min k plane desired  (zero based)
 *       maxk        = max k plane desired  (zero based)
 *       stepk       = k stride
 *
 *
 *  returns: Z_OK    if no problems
 *           Z_ERR   if an error
 *
 *  Notes:
 *  * It will not be called unless USERD_get_structured_reader_cinching
 *    indicates that this reader does structured cinching by returning
 *    a Z_OK.
 *
 *  * It will actually be called before each geom component and before
 *    each part variable - so if you are storing things locally, you should
 *    make this routine be able to quickly check whether anything needs
 *    updated or not.
 *
 *  * If the stride (step) does not hit right on the max, the last element
 *    in each direction will be shortened appropriately.
 *    For example, if a block had 0 to 12 in the i direction,
 *                 and the user specified min = 1
 *                                        max = 8
 *                                        step = 3
 *
 *             0   1   2   3   4   5   6   7   8   9   10  11  12
 *             |   |   |   |   |   |   |   |   |   |   |   |   |
 *
 *                 |           |           |   |
 *
 *      Namely, the coarser cell boundaries in this direction would be
 *              at 1, 4, 7, and 8
 *
 * IMPORTANT!!  If your reader will be used for structured auto-distribute,
 *              you must implement this feature, which includes this routine and
 *              USERD_get_structured_reader_cinching
 *----------------------------------------------------------------------*/
int
USERD_set_block_range_and_stride(int part_number,
```

```
int mini, int maxi, int stepi,
int minj, int maxj, int stepj,
int mink, int maxk, int stepk)
```

```
/*----------------------------------------------------------------------------
 USERD_set_extra_gui_data
 *                                          <optional> (Version 2.00 and later)
 *----------------------------------------------------------------------------
 *
 *  Receives the toggle, pulldown and field_text from enhanced GUI.
 *
 *  (IN) toggle values   TRUE = toggle checked
 *                       FALSE = toggle unchecked
 *                       Is num_Toggles long, as set in
 *                         USERD_get_extra_gui_numbers
 *
 *  (IN) pulldown value  from 0 to number of pulldown values
 *                       Is num_pulldowns long, as set in
 *                         USERD_get_extra_gui_numbers
 *
 *  (IN) field text      any text
 *                      '\0' if inactivated or nothing entered
 *                       Is num_fields by Z_LEN_GUI_FIELD_STR, as set in
 *                       USERD_get_extra_gui_numbers
 *
 *  Notes:
 *  This routine is called when the library is permanently
 *   loaded to the EnSight session, so define your globals
 *   in this and later routines.
 *
 *  It's up to you to change your reader behavior according to
 *    user entries!
 * ------------------------------------------------------------- */
void
USERD_set_extra_gui_data(int *toggle,
                         int *pulldown,
                         char **field_text)
```

```
/*----------------------------------------------------------------------
  USERD_set_filename_button_labels
 *                                <optional> (Version 2.07 and later)
 *----------------------------------------------------------------------
 *
 *   Returns the labels that the EnSight GUI will place on the buttons
 *   in the Data Reader/Open dialog for Geometry and Results
 *
 *   (OUT) filename_label_1  = Label for the first button
 *                             (Z_MAX_USERD_NAME long)
 *                                (generally the geom file)
 *
 *   (OUT) filename_label_2  = Label for the second button
 *                            (Z_MAX_USERD_NAME long)
 *                                (generally the results file)
 *                             Not needed (so can be null) if two_fields
 *                             is FALSE in USERD_get_name_of_reader
 *
 *  Notes:
 *----------------------------------------------------------------------*/
void
USERD_set_filename_button_labels(char filename_label_1[Z_MAX_USERD_NAME],
                                 char filename_label_2[Z_MAX_USERD_NAME])
```

```
/*----------------------------------------------------------------------
 USERD_set_filenames
 *                                              (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Receives the geometry and result filenames entered in the data
 *   dialog.  The user written code will have to store and use these
 *   as needed.  The user written code must manage its own files!!
 *
 *  (IN) filename_1   = the filename entered into the geometry
 *                      field of the data dialog.
 *
 *  (IN) filename_2   = the filename entered into the result
 *                      field of the data dialog.
 *                      (If the two_fields flag in USERD_get_name_of_reader
 *                       is FALSE, this will be null string)
 *
 *                       If two_fields is TRUE, this is the
 *                       mandatory results file entered
 *                       into the result field of the data dialog.
 *
 *                       If two_fields is -1, then this contains
 *                       optional text (filenames, modifiers, etc.)
 *                       that can be parsed and used to modify
 *                       reader
 *
 *  (IN) the_path     = the path info from the data dialog.
 *                      Note: filename_1 and filename_2 have already
 *                            had the path prepended to them.  This
 *                            is provided in case it is needed for
 *                            filenames contained in one of the files
 *
 *  (IN) swapbytes    = TRUE if should swap bytes when reading data.
 *                    = FALSE normally
 *
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * Since you must manage everything from the input that is entered in
 *    these data dialog fields, this is an important routine!
 *
 *  * Since you manage these files, they can be whatever.  Perhaps
 *    you will use only one, and have references to everything else
 *    you need within it, like EnSight's case file does.
 *----------------------------------------------------------------------*/
int
USERD_set_filenames(char filename_1[],
                    char filename_2[],
                    char the_path[],
                    int swapbytes)
```

```
/*----------------------------------------------------------------------
 *                            <optional>     (Version 2.05 and later)
 *----------------------------------------------------------------------
 *
 *   Informs the reader that the time currently set is the right side of a time
 *   span used for variable interpolation between time steps
 *
 *  Notes:
 *  * Applies to Current_time_step
 *
 *  * This is called just before USERD_get_var_by_component
 *
 *  * This information is only needed if your reader must do its own
 *    interpolation along a variable timeline.  This can occur when rigidbody
 *    information has its own timeline, which is sent to EnSight as the
 *    controlling time line, but the variables have a different timeline
 *    known only to the reader.
 *----------------------------------------------------------------------*/
void
USERD_set_right_side( void )
```

USERD_set_right_side

```
/*----------------------------------------------------------------------
 USERD_set_server_number
 *                                            (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Receives the server number of how many total servers.
 *
 *  (IN) cur_serv    = the current server.
 *
 *  (IN) tot_servs   = the total number of servers.
 *
 *  Notes:
 *  * Only useful if your user defined reader is being used with EnSight's
 *    Server-of-Server capability.  And even then, it may or may not be
 *    something that you can take advantage of.  If your data is already
 *    partitioned in some manner, such that you can access the proper
 *    portions using this information.
 *
 *    For all non-SOS uses, this will simply be 1 of 1
 *
 *  * Really just a dummy for this reader - we don't need to use it.
 *----------------------------------------------------------------------*/
void
USERD_set_server_number(int cur_serv,
                        int tot_servs)
```

```
/*----------------------------------------------------------------------
 USERD_set_time_set_and_step
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Set the current time step in the desired timeset.  All functions that
 *   need time, and that do not explicitly pass it in, will use the timeset
 *   and step set by this routine, if needed.
 *
 *  (IN) timeset_number  = the timeset number (1 based).
 *
 *                        For example:  If USERD_get_number_of_timesets
 *                                      returns 2, the valid timeset_number's
 *                                      would be 1 and 2.
 *
 *  (IN) time_step - The current time step (0 to Num_time_steps[timeset_number])
 *
 *  Notes:
 *  * Current_time_step and Current_timeset would be set here
 *----------------------------------------------------------------------*/
void
USERD_set_time_set_and_step(int timeset_number,
                            int time_step)
```

```
/*----------------------------------------------------------------------------
  USERD_set_var_extract_gui_data
 *                                          <optional> (Version 2.05 and later)
 *----------------------------------------------------------------------------
 *
 *  Receives the toggle, pulldown and field_text from var extract input.
 *
 *  (IN) toggle values   TRUE = toggle checked
 *                       FALSE = toggle unchecked
 *                       Is num_Toggles long, as set in
 *                         USERD_get_var_extract_gui_numbers
 *
 *  (IN) pulldown value  from 0 to number of pulldown values
 *                       Is num_pulldowns long, as set in
 *                         USERD_get_var_extract_gui_numbers
 *
 *  (IN) field text      any text
 *                      '\0' if inactivated or nothing entered
 *                       Is num_fields by Z_LEN_GUI_FIELD_STR, as set in
 *                       USERD_get_var_extract_gui_numbers
 *
 *  Notes:
 *  This routine is called when the library is permanently
 *   loaded to the EnSight session, so define your globals
 *   in this and later routines.
 *
 *  It's up to you to change your reader behavior according to
 *    user entries!
 * ------------------------------------------------------------- */
void
USERD_set_var_extract_gui_data(int *toggle,
                               int *pulldown,
                               char **field_text)
```

```
/*----------------------------------------------------------------------
 USERD_size_matf_data
 *                                              (Version 2.03 and later)
 *----------------------------------------------------------------------
 *
 *   Get the length of the material id list, mixed-material id list, or
 *   mixed-material values list for the given material set and part (and
 *   element type if material id list)
 *
 *  (IN)  set_index         = the material set index (zero based)
 *
 *  (IN)  part_id           = the part number desired
 *
 *  (IN)  wtyp              = the element type (used for Z_MAT_INDEX only)
 *
 *                                Z_POINT    node point element
 *                                Z_BAR02    2 node bar
 *                                Z_BAR03    3 node bar
 *                                Z_TRI03    3 node triangle
 *                                Z_TRI06    6 node triangle
 *                                Z_QUA04    4 node quad
 *                                Z_QUA08    8 node quad
 *                                Z_TET04    4 node tetrahedron
 *                                Z_TET10   10 node tetrahedron
 *                                Z_PYR05    5 node pyramid
 *                                Z_PYR13   13 node pyramid
 *                                Z_PEN06    6 node pentahedron
 *                                Z_PEN15   15 node pentahedron
 *                                Z_HEX08    8 node hexahedron
 *                                Z_HEX20   20 node hexahedron
 *                                Z_NSIDED  nsided polygon
 *                                Z_NFACED  nfaced polyhedron
 *
 *                                Z_G_POINT    ghost node point element
 *                                Z_G_BAR02    2 node ghost bar
 *                                Z_G_BAR03    3 node ghost bar
 *                                Z_G_TRI03    3 node ghost triangle
 *                                Z_G_TRI06    6 node ghost triangle
 *                                Z_G_QUA04    4 node ghost quad
 *                                Z_G_QUA08    8 node ghost quad
 *                                Z_G_TET04    4 node ghost tetrahedron
 *                                Z_G_TET10   10 node ghost tetrahedron
 *                                Z_G_PYR05    5 node ghost pyramid
 *                                Z_G_PYR13   13 node ghost pyramid
 *                                Z_G_PEN06    6 node ghost pentahedron
 *                                Z_G_PEN15   15 node ghost pentahedron
 *                                Z_G_HEX08    8 node ghost hexahedron
 *                                Z_G_HEX20   20 node ghost hexahedron
 *                                Z_G_NSIDED  ghost nsided polygon
 *                                Z_G_NFACED  ghost nfaced polyhedron
 *
 *  (IN)  mat_type          = Z_MAT_INDEX for material ids list
 *                            Z_MIX_INDEX for mixed-material ids list
 *                            Z_MIX_VALUE for mixed-material values list
 *                            Z_SPE_VALUE for material species values
 *
 *  (OUT) matf_size         = the length of the material id list, or
 *                            mixed-material id list, or
 *                            mixed-material values list
 *                            for the given material set and part number
 *                            (and element type if Z_MAT_INDEX)
```

```
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * See USERD_get_number_of_material_sets header for explanatory example
 *  * Will not be called if Num_material_sets is zero, or
 *     Num_materials[set_index] is zero
 *  * This function is used only if USERD_get_matf_type returns
 *     Z_MISET_VIA_SPARSE_MIX.
 *  * This function is NOT usAed if USERD_get_matf_type returns
 *     Z_MISET_VIA_ESCAL_VARS (and thus is incompatible with
 *     USERD_get_matf_escalars_desc)
 *-----------------------------------------------------------------------*/
int
USERD_size_matf_data( int set_index,
                      int part_id,
                      int wtyp,
                      int mat_type,
                      int *matf_size)
```

```
/*----------------------------------------------------------------------
  USERD_stop_part_building
 *                                          (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *   Called when part builder is closed for USERD, can be used to clean
 *   up memory, etc that was only needed during the part building process.
 *----------------------------------------------------------------------*/
void
USERD_stop_part_building( void )
```

```
/*----------------------------------------------------------------------
 USERD_use_periodic_ghosts
 *                                <optional>   (Version 2.00 and later)
 *----------------------------------------------------------------------
 *
 *  The existence of the this as well as the following routines, implies to
 *    EnSight that the reader has the capability to read periodic models, and
 *    a return value of TRUE from this routine tells EnSight that at least one
 *    part has periodic symmetry faces and the following routines should be
 *    called.  A return value of FALSE tells EnSight that no parts have
 *    periodic symmetry faces and not to call the remaining routines.
 *
 *      USERD_get_periodic_ghosts_num_symmetry_faces
 *
 *      and for unstructured parts:
 *      ---------------------------
 *      USERD_get_periodic_ghosts_num_pairs
 *      USERD_get_periodic_ghosts_pairs.
 *
 *      and for structured parts:
 *      -------------------------
 *      USERD_get_periodic_ghosts_structured_face_info
 *
 *
 *  returns: TRUE    If there is at least one part that needs to
 *                   use the node pairing information (for unstructured parts),
 *                   or the face ijk information (for structured parts), or
 *                   supplied transformation matrices, to augment the initial
 *                   periodic instance with the proper ghosts across the
 *                   symmetry face(s).
 *
 *           FALSE   if no parts have periodic faces and don't call the other
 *                   periodic routines.
 *
 *
 *   Simple Examples:
 *
 *   periodic translation of 3 units in x model
 *   ==========================================
 *               y
 *                13        14        15        16
 *               *---------*---------*---------*
 *               |         |         |         |
 *               |    6    |    7    |    8    |
 *               |9        |10       |11       |12
 *               *---------*---------*---------*
 *               |         |         |         |
 *     face 0    |    3    |    4    |    5    |    face 1
 *               |5        |6        |7        |8
 *               *---------*---------*---------*
 *               |         |         |         |
 *               |    0    |    1    |    2    |
 *               |1        |2        |3        |4
 *               *---------*---------*---------*  x
 *
 *              <===== translation direction =====>
 *
 *               y
 *      15        13        14        15        16        14
 *     *- - - - -*---------*---------*---------*- - - - -*
 *     .  ghost  |         |         |         |  ghost  .
```

```
*     .    8    |    6    |    7    |    8    |    6    .
*     .11       |9        |10       |11       |12       .10
*     *- - - - -*---------*---------*---------*- - - - -*
*     .  ghost  |         |         |         |  ghost  .
*     .    5    |    3    |    4    |    5    |    3    .
*     .7        |5        |6        |7        |8        .6
*     *- - - - -*---------*---------*---------*- - - - -*
*     .  ghost  |         |         |         |  ghost  .
*     .    2    |    0    |    1    |    2    |    0    .
*     .3        |1        |2        |3        |4        .2
*     *- - - - -*---------*---------*---------*- - - - -*  x
*
*  num_symmetry_faces = 2;
*
*    face 0
*    ------
*     num_pairs = 4
*     pairs[0] = 1  node 1 moves to node 4 location
*     pairs[1] = 4
*     pairs[2] = 5  node 5 moves to node 8 location
*     pairs[3] = 8
*     pairs[4] = 9  node 9 moves to node 12 location
*     pairs[5] = 12
*     pairs[6] = 13 node 13 moves to node 16 location
*     pairs[7] = 16
*
*    face 1
*    ------
*     num_pairs = 4
*     pairs[0] = 4  node 4 moves to node 1 location
*     pairs[1] = 1
*     pairs[2] = 8  node 8 moves to node 5 location
*     pairs[3] = 5
*     pairs[4] = 12 node 12 moves to node 9 location
*     pairs[5] = 9
*     pairs[6] = 16 node 16 moves to node 13 location
*     pairs[7] = 13
*
*-------------------------------------------------------------------------
*   periodic rotation of 90 degrees about z axis
*   ============================================
*               y
*                13        14        15        16
*               *---------*---------*---------*
*               |         |         |         |
*               |    6    |    7    |    8    |
*               |9        |10       |11       |12
*               *---------*---------*---------*
*               |         |         |         |
*    face 1     |    3    |    4    |    5    |
*               |5        |6        |7        |8
*               *---------*---------*---------*
*               |         |         |         |
*               |    0    |    1    |    2    |
*               |1        |2        |3        |4
*               *---------*---------*---------*  x
*              /            face 0
*             /
*            -- rotates in plane about node 1
*
*
```

```
*               y
*       8        13        14        15        16
*     *- - - - -*---------*---------*---------*
*     .  ghost  |         |         |         |
*     .    2    |    6    |    7    |    8    |
*     .7        |9        |10       |11       |12
*     *- - - - -*---------*---------*---------*
*     .  ghost  |         |         |         |
*     .    1    |    3    |    4    |    5    |
*     .6        |5        |6        |7        |8
*     *- - - - -*---------*---------*---------*
*     .  ghost  |         |         |         |
*     .    0    |    0    |    1    |    2    |
*     .5        |1        |2        |3        |4
*     *- - - - -*---------*---------*---------*
*               .  ghost  .  ghost  .  ghost  .
*               .    0    .    3    .    6    .
*               .2        .6        .10       .14
*               *- - - - -*- - - - -*- - - - -*
*
*  num_symmetry_faces = 2;
*
*    face 0
*    ------
*     num_pairs = 4
*     pairs[0] = 1  node 1 rotates to node 1 location
*     pairs[1] = 1
*     pairs[2] = 2  node 2 rotates to node 5 location
*     pairs[3] = 5
*     pairs[4] = 3  node 3 rotates to node 9 location
*     pairs[5] = 9
*     pairs[6] = 4  node 4 rotates to node 13 location
*     pairs[7] = 13
*
*    face 1
*    ------
*     num_pairs = 4
*     pairs[0] = 1  node 1 rotates to node 1 location
*     pairs[1] = 1
*     pairs[2] = 5  node 5 rotates to node 2 location
*     pairs[3] = 2
*     pairs[4] = 9  node 9 rotates to node 3 location
*     pairs[5] = 3
*     pairs[6] = 13 node 13 rotates to node 4 location
*     pairs[7] = 4
*
*
*-------------------------------------------------------------------------
*   mirror in x
*   ===========
*               y
*                13        14        15        16
*               *---------*---------*---------*
*               |         |         |         |
*               |    6    |    7    |    8    |
*               |9        |10       |11       |12
*               *---------*---------*---------*
*               |         |         |         |
*     face 0    |    3    |    4    |    5    |
*               |5        |6        |7        |8
*               *---------*---------*---------*
```

```
 *               |         |         |         |
 *               |    0    |    1    |    2    |
 *               |1        |2        |3        |4
 *               *---------*---------*---------*  x
 *    <===  mirror
 *
 *               y
 *      14        13        14        15        16
 *     *- - - - -*---------*---------*---------*
 *     .  ghost  |         |         |         |
 *     .    6    |    6    |    7    |    8    |
 *     .10       |9        |10       |11       |12
 *     *- - - - -*---------*---------*---------*
 *     .  ghost  |         |         |         |
 *     .    3    |    3    |    4    |    5    |
 *     .6        |5        |6        |7        |8
 *     *- - - - -*---------*---------*---------*
 *     .  ghost  |         |         |         |
 *     .    0    |    0    |    1    |    2    |
 *     .2        |1        |2        |3        |4
 *     *- - - - -*---------*---------*---------*
 *
 *  num_symmetry_faces = 1;
 *
 *    face 0
 *    ------
 *     num_pairs = 4
 *     pairs[0] = 1  node 1 reflects about node 1
 *     pairs[1] = 1
 *     pairs[2] = 5  node 5 reflects about node 5
 *     pairs[3] = 5
 *     pairs[4] = 9  node 9 reflects about node 9
 *     pairs[5] = 9
 *     pairs[6] = 13 node 13 reflects about node 13
 *     pairs[7] = 13
 *-------------------------------------------------------------------------*/
int
USERD_use_periodic_ghosts( void )
```

2.5 Converting a 1.0 API Reader to a 2.0 API READER

If you have an existing 1.0 API Reader and you desire to convert it to a 2.0 API reader, to take advantage of new capabilities, or the improved efficiency, the following may be helpful.

First the Good News!

The following routines were identical in both API's at the time that the 2.0 API was produced.

USERD_bkup
USERD_get_block_coords_by_component
USERD_get_block_iblanking
USERD_get_changing_geometry_status
USERD_get_dataset_query_file_info
USERD_get_element_label_status
USERD_get_name_of_reader
USERD_get_node_label_status
USERD_get_number_of_files_in_dataset
USERD_get_number_of_model_parts
USERD_get_number_of_variables
USERD_set_filenames
USERD_stop_part_building

Second, pretty Good News!

The following routines have minor changes, namely a slight name change and the addition of arguments related to complex data, constant type, or self contained parts vs global coords.

(Note, the name changes are needed so both API's can exist together)

The arguments must be added, but depending on your situation, many might simply be place holders.

A) Changes related to imaginary flag for complex data

If you don't deal with complex variables, simply add this flag to your argument list and ignore its value.

API 1.0	API 2.0

`USERD_get_constant_value`
`(`
`  int which var`
 	
`)`
	`USERD_get_constant_val`
	`(`
	`  int which_var,`
	`  int imag_data`
	`)`

`USERD_get_description_lines`
`(`
`  int which_type,`
`  int which_var,`
 	
`  char line1[Z_BUFL],`
`  char line2[Z_BUFL]`
`)`
	`USERD_get_descrip_lines`
	`(`
	`  int which_type,`
	`  int which_var,`
	`  int imag_data,`
	`  char line1[Z_BUFL],`
	`  char line2[Z_BUFL]`
	`)`

USERD_get_variable_value_at_specific
(
  int which_var,
  int which_node_or_elem,
  int which_part,
  int which_elem_type,
  int time_dtep,
  float values[3]
 	
)
	USERD_get_var_value_at_specific
	(
	  int which_var,
	  int which_node_or_elem,
	  int which_part,
	  int which_elem_type,
	  int time_dtep,
	  float values[3],
	  int imag_data
	)

B) Changes related to complex data info, and constant type (and some of the multiple timeset support).

If you don't deal with complex variables, simply add the arguments for var_complex, var_ifilename, and var_freq and assign var_complex to be FALSE.

The argument var_contran needs to be added, and set appropriately if you have constant variables, to indicate if the constant variable is fixed for all time or varies over time.

The argument var_timeset needs to be added, and set appropriately.

API 1.0	API 2.0

USERD_get_variable_info
(
  char **var_description,
  char **var_filename,
  int *var_type,
  int *var_classify
 	
)
	USERD_get_gold_variable_info
	(
	  char **var_description,
	  char **var_filename,
	  int *var_type,
	  int *var_classify,
	  int *var_complex,
	  char **var_ifilename,
	  float *var_freq,
	  int *var_contran,
	  int *var_timeset
	)

C) Changes related to self contained part coordinates.

The number_of_nodes argument needs to be added and set for each part. **This one is critical for you to do.**

API 1.0	API 2.0

USERD_get_part_build_info
(
  int *part_id,
  int *part_types,
  char *part_description[Z_BUFL],
 	
  int *number_of_elements[Z_MAXTYPE],
  int *ijk_dimensions[3],
  int *iblanking_options[6]
)
	USERD_get_gold_part_build_info
	(
	  int *part_id,
	  int *part_types,
	  char *part_description[Z_BUFL],
	  int *number_of_nodes,
	  int *number_of_elements[Z_MAXTYPE],
	  int *ijk_dimensions[9],
	  int *iblanking_options[6]
	)

D) Changes related to multiple timeset support. The timeset_number argument needs to be added for the following three routines. The multiple timeset support also includes the change in B) above for USERD_get_gold_variable_info and the three new routines in the next section.	
API 1.0	API 2.0

USERD_get_number_of_time_steps
(
  void
)
	USERD_get_num_of_time_steps
	(
	  int timeset_number
	)

USERD_get_solution_times
(
 	
  float *solution_times
)
	USERD_get_sol_times
	(
	  int timeset_number,
	  float *solution_times
	)

USERD_set_time_step
(
 	
  int time_step
}
	USERD_set_time_set_and_step
	(
	  int timeset_number,
	  int time_step
	)

Third, deleted and new routines. (Here is where the work lies)

Several old routines are gone. You will have to create the new routines that replace them. I think you will find in most cases that your old routines will form the basis of the new routines, and that it isn't too difficult to provide the information in the new way.

See Detailed Specifications in this chapter for the needed information on these new routines.

API 1.0	API 2.0
These routines: USERD_get_block_scalar_values USERD_get_block_vector_values_by_component USERD_get_scalar_values USERD_get_vector_values	**replaced by the single routine:** USERD_get_var_by_component
These global coordinate routines: USERD_get_global_coords USERD_get_global_node_ids USERD_get_number_of_global_nodes	**replaced by part coord routines:** USERD_get_part_coords USERD_get_part_node_ids
These par connectivity routines: USERD_get_element_connectivities_for_part USERD_get_element_ids_for_part	**replaced by part by type routines:** USERD_get_part_elements_by_type USERD_get_part_element_ids_by_type
	These are new routines: (Can be a dummy) -> USERD_exit_routine (Can be a dummy) -> USERD_get_model_extents (Required) -> USERD_get_reader_version

	multiple timeset related: (Required) -> USERD_get_number_of_timesets (Required) -> USERD_get_timeset_description (Required) -> USERD_get_geom_timeset_number
	border provided by the reader option: (Required) -> USERD_get_border_availability (Can be a dummy) -> USERD_get_border_elements_by_type
	transient model allocation efficiency: (Can be a dummy) -> USERD_get_maxsize_info
	possible use with Server-of-Servers: (Can be a dummy) -> USERD_set_server_number
	Required routines added after version 2.00 (Many can be dummy routines, depending on features needed): USERD_get_block_ghost_flags USERD_get_ghosts_in_block_flag USERD_get_ghosts_in_model_flag USERD_get_matf_set_info USERD_get_matf_var_info USERD_get_number_of_material_sets USERD_get_number_of_materials USERD_load_matf_data USERD_size_matf_data USERD_get_nfaced_conn USERD_get_nfaced_nodes_per_face USERD_get_nsided_conn USERD_get_uns_failed_params USERD_get_matsp_info USERD_get_number_of_species USERD_rigidbody_existence USERD_rigidbody_values USERD_get_structured_reader_cinching USERD_set_block_range_and_stride
	Also note the various optional routines which can be in the 2.0 API. See the Routine History for an easy identification of these routines

3 User Defined Writer API

Users can write User-Defined Writers (UDW) to generate arbitrary data files for EnSight parts and variables. The EnSight server provides a UDW API that can be used to query the currently selected parts in the EnSight client part list. The UDW API includes methods to get, for example, node coordinates, element connectivity, ids, variable values, and time information. A UDW can call any of the methods as it wishes and create a data file(s) in any format desired. Additionally, the UDW dialog in the EnSight client has a Parameter field that provides a mechanism for passing user specified options to the UDW.

What Information Can Be Provided By The API?

Which parts are available to the UDW?	All parts currently selected in the Main Parts List (except those indicated below)
Where are the available parts located?	On the EnSight server
Which parts are unavailable to the UDW?	Any client-based parts: contours vector arrows particle traces profiles

Example Writers

Several example User-Defined Writers (including source code, Makefile, and shared library) are included to demonstrate this capability. The easiest way to get started is to copy the whole directory of a simple writer, such as the Flatfile writer, then change it's name, modify the Makefile, set the environment variable `ENSIGHT9_UDW`, and make it. Once you have it made, start EnSight with the following option:

```
ensight100 -writerdbg
```

to verify that your writer is loading properly. Once loading properly, use print statements to aid in debugging. Alternatively, attach a debugger to the ensight100.server at run time and set breakpoints within the methods of the UDW.

The **Flatfile UDW** is designed to demonstrate the output of selected part nodal data (coordinates & IDs) as well as active variable values (scalar and/or vector only) in a comma delimited format easily imported into other applications. If any of the keywords 'ANSYS' or 'force' or 'body' is entered into the Parameter field of the EnSight client UDW dialog, then the Flatfile UDW will output an ANSYS body force file. If keywords 'SSCALE #', 'VSCALE #' and/or 'GSCALE #' are entered then the scalars, vectors, and/or geometry are each scaled by the number following the keyword when written out to the file.

The **HDF 5.0 UDW** is designed to write out selected parts and their corresponding active variables using the HDF 5.0 API which is compatible with the EnSight HDF User-Defined Reader. The HDF writer ignores the Parameter field. The HDF 5.0 writer illustrates most of the routines available to retrieve data from EnSight.

The **Case (Gold) Lite UDW** is provided to demonstrate how to exercise most of the API and output a subset of the Case (Gold) format. Complex numbers and the custom Gold format are not supported in this writer. The Case (Gold) writer ignores the Parameter field. While the writer is not provided as a

prebuilt library, the source code and Makefile are provided.

The **STL UDW** is provided to write out the border geometry in the form of triangular 2D elements for the selected part(s) at the beginning timestep. The end time and the step time are ignored. The STL format does not support multiple parts in a single binary file, but does support multiple parts in a single ASCII file. Therefore, if multiple parts are selected and ascii is checked, the STL writer outputs an ascii file with the border of each of the parts. If multiple parts are selected and binary is checked, the STL writer outputs a binary file containing a single border of the multiple parts. The STL writer ignores the Parameter field.

3.1 Directions For Writing Your Own UDW

1. Create a directory where your writer will be located, for example

   ```
   $CEI_HOME/ensight100/src/writers/mywriter/
   ```

2. Several example writers are provided which have source code and a Makefile.
   For example, look at the flatfile format, an ASCII comma delimited writer.

   ```
   cd $CEI_HOME/ensight100/src/writers/flatfile
   ```

   Notice that there are several files in this directory.
   a. libuserd_write.c - The writer code
   b. Makefile - makefile
   c. README - specific directions for using this writer

3. Copy these files into your directory:

   ```
   $CEI_HOME/ensight100/src/writers/mywriter/
   ```

   a. The Makefile should be used to compile and link a shared library.
   Edit the Makefile so that it names the shared library properly.

   b. Edit the C file.

   i. USERD_writer_get_name - give the writer a name. Ignore the other returned variable.

   ii. USERD_writer_get_writer_version - give the writer a version so you can use version control in the future revisions

   iii. USERD_writer_write_geom - The UDW routine called by EnSight. What you do in here is up to you; but basically you can open a file for writing, call a bunch of methods (listed below) to get the data of interest, write data into a file, and return an error status code to EnSight.

   This method has the following arguments:

a. char full_fname[Z_MAXFILENP]	- (IN)	file name requested by the user from the GUI
b. int lis_parts[Z_MAXPART]	- (IN)	list of parts selected by the user in EnSight
c. int num_parts	- (IN)	number of selected parts
d. int do_binary	- (IN)	TRUE if writing binary file FALSE if writing ascii file
e. float max_fsize_mb	- (IN)	maximum file size value for this machine
f. int combined	- (IN)	TRUE if user requests single file output FALSE if user allows multiple file output
g. float *timestep_vals	- (IN)	array of time step values
h. int ntime_steps	- (IN)	number of time steps
i. char text_input[UDW_STRSIZE]	- (IN)	string entered from GUI by user can be used to input commands to modify writer behavior
j. int *error_flag	- (OUT)	Return from writer Z_ERR if a problem Z_EN_ERR_NONE if no problem

   c. Edit the README file

   Since each writer can operate however it wishes, document any constraints, expectations, limitations, user specified parameters, etc. here. Given this is likely the only documentation available for the writer, give enough details about it for both end-users and future maintainers.

d. Set the UDW environment variable:

```
setenv ENSIGHT9_UDW $CEI_HOME/ensight100/src/writers/mywriter/
```

EnSight will first look in `$ENSIGHT9_UDW` and load the writer library. Next, EnSight will then look in `$CEI_HOME/ensight100/machines/$CEI_ARCH/lib_writers/`. If duplicate writers are found, they will only be loaded once.

e. Make your library and fix all compile errors.

f. Run the EnSight using a manual connection and specify the command line option -writerdbg to the server to verify that it is loading the UDW correctly at runtime:

`ensight100.server -writerdbg` (or for Windows `ensight100_server -writerdbg`)

g. Compile in UDW debug output to track the progress of runtime loading of the writer and proper operation.

Topical List Of User-Defined Writer API Methods

The following is a topical list of the User-defined Writer API methods along with a brief description of each. These methods should be called from within the UDW's USERD_writer_write_geom() method to retrieve data from EnSight.

GENERAL INFO	
USERD_writer_part_verify	a. Good part if at least one part > 0 b. Is at least one part created geometry? (Not discrete and not model part) c. Do I need to write model geometry? (not discrete particle type) d. Do I need to write measured geometry? (discrete particle type)
USERD_get_undef_ptr	Checks each element type for any undefined variable values
USERD_get_num_time_steps	Return the time-set num steps index based on var_index and meas_data
USERD_writer_get_variable_transient	Returns variable static (0) or transient (1)
USERD_writer_get_var_type	TRUE if model vars = static && measured vars = constant
VARIABLE INFO	
USERD_writer_get_undef_val	Echoes the EnSight undefined value
USERD_writer_get_exist_active	Does var exist and is it active?
USERD_writer_get_variable_info	Gets variable descriptors source, complex, type, descrip, vref, freq, parent
USERD_writer_get_per_elem_node	Gets per element or per node flags, any undefined flag
USERD_writer_get_number_of_variables	Returns number of variables
USERD_writer_get_measured_vector_var_val	Returns vector of measured variable values
USERD_writer_get_part_variable_status	T or F, does part have variable(s)?
USERD_writer_get_static_const_value	A variable's current constant value
USERD_writer_get_component_vector_var_val	Returns vector of variable value

PART INFO	
USERD_writer_get_part_info	Gets part_type, num elems, descrip, struct_flag
USERD_writer_get_part_struct_unstruct	Returns either structured or unstructured
USERD_writer_get_changing_measured_geometry_flag	Measured geom change status
USERD_writer_get_changing_model_geometry_flag	Coord, connect change status
USERD_writer_get_structured_data	Gets values assoc w/ struct data part_ijk_num, iblank nf, ghost flag, cell type
USERD_writer_get_structured_data_ijk	Gets values assoc w/ struct data part_ijk_num, iblank nf, ghost flag, cell type
USERD_writer_get_structured_cell_type	Gets only the cell type for structured data
NODAL INFO	
USERD_writer_get_node_label_status	Does MODEL have node labels?
USERD_writer_get_part_node_label_status	Does PART have node labels? (T or F)
USERD_writer_get_part_coords	Gets part x, y, & z coordinates in 2d format
USERD_writer_get_part_coords_vector	Gets part x, y, & z coordinates in vector format
USERD_writer_get_part_node_id	Gets part node ids
ELEMENT INFO	
USERD_writer_get_element_label_status	Does MODEL have element labels?
USERD_writer_get_part_element_label_status	Does PART have element labels? (T or F)
USERD_writer_get_part_elem_id	Returns array of element id's
USERD_writer_get_part_elem_id_per_type	Returns array of element id's per elem type
USERD_writer_get_eletype_string	Get the string describing that element type
USERD_writer_get_element_connectivities_for_part	Elem connectivity vector
USERD_writer_get_element_connectivities_for_part_simple	Elem connectivity by elem type
USERD_get_Nfaced_size	Total vector length, index of first connectivity val, max conn size, total # faces
USERD_get_Nfaced_vector	Gets vector of connectivity data
TIME	
USERD_writer_validate_time_step	Validates the current Ensight time step for multiple scales
USERD_writer_get_original_time	Gets client time which is the original time.
USERD_writer_set_current_time	Sets the time
OTHER	
USERD_get_titles	Gets the two model title description lines
USERD_writer_get_ensight_release	Get Ensight release letter as string
USERD_writer_get_ensight_version	Get Ensight version number as string

USERD_writer_whatis_machine_byte_order	Current machine is big- or little-endian
SIZING	
USERD_writer_get_var_max_sect_vals	Loop thru parts & find max # variables of node or elem type
USERD_get_modl_geo_max_node_size	Loop thru parts & find max # of nodes
USERD_get_modl_geo_max_conn_size	Loop thru parts and find max connectivity size
USERD_get_current_model_extents	Either assign or compute model ranges
SPECIALIZED	
USERD_writer_get_part_coords_per_elem	Gets part coordinates fro TRI & QUA elems
USERD_writer_get_part_coords_per_elem_border	Gets part border coords for TRI & QUA
USERD_run_border	Calls create_border and finds boundary of part

3.2 Routine Detail Specifications

Include files:

The following header file is required in any file containing these library routines.

```
#include "../extern/global_extern_w.h"
```

And for windows, the following is referenced from within:

```
#include "global_extern_w_dispatch.h"
```

Global Define:

The following should be defined in your writer code.

```
#define USERD_WRITER_GLOBALS
```

```
/*-----------------------------------------------------------------------*/
```

USERD_writer_get_name

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*   Gets the name of the writer, so gui can list as a writer            */
/*                                                                       */
/*  (OUT) writer_name          = the name of the writer (data format)    */
/*                              (max length is Z_MAX_USERD_NAME, which   */
/*                               is 20)                                  */
/*                                                                       */
/*  (OUT) *two_fields          = FALSE if only one data field is         */
/*                                      required.                        */
/*                                TRUE if two data fields required       */
/*                                                                       */
/*  returns: Z_OK  if successful                                         */
/*           Z_ERR if not successful                                     */
/*                                                                       */
/*  Notes:                                                               */
/*  * Always called.  Provide a name for your custom writer format       */
/*                                                                       */
/*  * If you don't want a custom writer to show up in the data dialog    */
/*    choices, return a name of "No Custom"                              */
/*-----------------------------------------------------------------------*/
int
USERD_writer_get_name(char writer_name[Z_MAX_USERD_NAME],
                      int *two_fields)
```

```
/*----------------------------------------------------------------------
```

USERD_writer_get_writer_version

```
 *----------------------------------------------------------------------
 *
 *   Gets the release string for the writer.
 *
 *   This release string is a free-format string.
 *   It is used for version control and backwards compatibility.
 *   It is useful to increment
 *   the release number/letter to indicate a change in the writer.
 *   The given string will simply be output by the EnSight server
 *   when the writer is selected.
 *
 *  (OUT) release_number       = the release number of the writer
 *                              (max length is Z_MAX_USERD_NAME, which
 *                               is 20)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *    Called when the writer is selected for use.
 *    called by USERD_writer routines
 *----------------------------------------------------------------------*/
int
USERD_writer_get_writer_version(char version_number[Z_MAX_USERD_NAME])
```

USERD_writer_write_geom

```
/*----------------------------------------------------------------------
 *----------------------------------------------------------------------
 *
 *  Write user specified data for selected parts and active variables.
 *
 *  (IN) char full_fname[Z_MAXFILENP] = file name requested by the user from the GUI
 *  (IN) int lis_parts[Z_MAXPART]     = list of parts selected by the user in EnSight
 *  (IN) int num_parts                = number of selected parts
 *  (IN) int do_binary                = TRUE  if writing binary file
 *                                      FALSE if writing ascii file
 *  (IN) float max_fsize_mb           = maximum file size value for this machine
 *  (IN) int combined                 = TRUE if user requests single file output
 *                                      FALSE if user allows multiple file output
 *  (IN) float *timestep_vals         = array of time step values
 *  (IN) int ntime_steps              = number of time steps
 *  (IN) char text_input[UDW_STRSIZE] = string entered from GUI by user can be used to
 *                                      input commands to modify writer behavior
 *  (OUT) int *error_flag             = Return from writer
 *                                          Z_ERR if a problem
 *                                          Z_EN_ERR_NONE if no problem
 *----------------------------------------------------------------------*/
void
USERD_writer_write_geom(char full_fname[Z_MAXFILENP],
                        int lis_parts[Z_MAXPART],
                        int num_parts,
                        int do_binary,
                        float max_fsize_mb,
                        int combined,
                        float *timestep_vals,
                        int ntime_steps,/
                        char text_input[ UDW_STRSIZE],
                        int *error_flag)
```

4 User Defined Math Functions

Users can write external variable calculator functions called User Defined Math Functions (UDMF) that can be dynamically loaded by EnSight. These functions appear in EnSight's calculator in the general function list and can be used just as any other calculator function to derive new variables.

Several examples of UDMFs can be found in the directory `$CEI_HOME/ensight100/src/math_functions/`. Please see these examples if you wish to create your own UDMFs.

When the EnSight server starts it will look in the following subdirectories for UDMF dynamic shared libraries:

```
./libudmf-devel.so (.sl) (.dll)
$ENSIGHT9_UDMF/libudmf-*.so (.sl) (.dll)
$CEI_HOME/ensight100/machines/$ENSIGHT9_ARCH/lib_udmf/libudmf-*.so (.sl) (.dll)
```

Depending on the server platform, the dynamic shared library must have the correct suffix for that platform (e.g. `.so, .sl, .dll`).

How the routines are invoked

Currently, when a UDMF is used in the EnSight calculator, it is invoked for each node in the specified part(s) if all the variables operated on for the specified part(s) are node centered. If all of the variables are element centered, then the UDMF is invoked for each element in the part(s). If the variables are a mix of node and element centered values, then the node centered values are automatically converted to element centered values and then the UDMF is invoked for each element using element centered variables.

Arguments and the return type for the UDMF can be either scalar or vector EnSight variables or constants.

Current Limitation

At this time, only variable quantities and constants can be passed into UDMFs. There is no mechanism for passing in either part geometry, neighboring variables, or other information.

4.1 Detailed Routine Specifications

Include files:

The following header file is required in any file containing these library routines.

```
#include "/udmf_extern.h"
```

```
/*----------------------------------------------------------------------
```

USERD_get_name_of_mf

```
 *----------------------------------------------------------------------
 *
 *   Gets the name of the math function, so gui can list as a calculator
 *   function.
 *
 *  (OUT) mf_name          = the name of the math function
 *                           (max length is UDMFSNAME, which is 64)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *----------------------------------------------------------------------*/
int
USERD_get_name_of_mf(char mf_name[UDMFSNAME])
```

```
/*----------------------------------------------------------------------
```
USERD_get_mf_version
```
 *----------------------------------------------------------------------
 *
 *   Gets the version number of the user defined math function supported API
 *
 *  (OUT) version_number       = the version number of the math function
 *                              (max length is UDMFSNAME, which is 64)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * version needs to be "1.000".
 *----------------------------------------------------------------------*/
int
USERD_get_mf_version(char version_number[UDMFSNAME]) {

  strcpy(version_number, "1.000");
  return(Z_OK);

}
```

USERD_get_nargs

```
/*----------------------------------------------------------------------
 *
 *   Gets the number of arguments needed by the function.
 *
 *  (OUT) nArgs       = the number of arguments
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *----------------------------------------------------------------------*/
int
USERD_get_nargs(int *nArgs) {

  *nArgs = 2;
  return(Z_OK);

}
```

```
/*-----------------------------------------------------------------------
```
USERD_get_meta_data
```
 *-----------------------------------------------------------------------
 *
 *   Get the function descriptions, argument types, and return type.
 *
 *  (OUT) listDescription = description shown in general function column
 *  (OUT) funcDescription = description shown in feedback window
 *  (OUT) argTypes = data types of arguments passed into USERD_evaluate
 *  (OUT) returnType = data type returned by function USERD_evaluate
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_meta_data(char listDescription[UDMFLNAME],
                    char funcDescription[UDMFLNAME],
                    int *argTypes, int *returnType) {

  strcpy(listDescription, "add2(part, scalar, scalar)");

  strcpy(funcDescription, "add2(any part(s), scalar, scalar)");

  argTypes[0] = UDMFSCL;
  argTypes[1] = UDMFSCL;

  *returnType = UDMFSCL;

  return(Z_OK);

}
```

USERD_evaluate

```
/*----------------------------------------------------------------------
 *
 *   Evaluate the function.
 *
 *  (OUT) args = pointers to arguments
 *  (OUT) undefined = boolean; true if return value is undefined.
 *  (OUT) value = returned value
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *----------------------------------------------------------------------*/
int
USERD_evaluate(void *args[], void *value, int *undefined) {

  float *result;
  float *arg1, *arg2;

  result = value;
  arg1 = args[0];
  arg2 = args[1];

  *result = *arg1 + *arg2;
  *undefined = 0;

  return(Z_OK);

}
```

4.2 Example

The following example simply adds two scalars. Other examples can be found in subdirectories of `$CEI_HOME/ensight100/src/math_functions/`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef WIN32
#include <unistd.h>
#endif

#include "../extern/udmf_extern.h"

/*-----------------------------------------------------------------------
 * USERD_get_name_of_mf
 *-----------------------------------------------------------------------
 *
 *   Gets the name of the math function, so gui can list as a calculator
 *   function.
 *
 *  (OUT) mf_name          = the name of the math function
 *                           (max length is UDMFSNAME, which is 64)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *-----------------------------------------------------------------------*/
int
USERD_get_name_of_mf(char mf_name[UDMFSNAME]) {
  memset(mf_name,'\0',UDMFSNAME);
  strcpy(mf_name, "addTwoScalars");
  return(Z_OK);
}

/*-----------------------------------------------------------------------
 * USERD_get_mf_version
 *-----------------------------------------------------------------------
 *
 *   Gets the version number of the user defined math function supported API
 *
 *  (OUT) version_number       = the version number of the math function
 *                               (max length is UDMFSNAME, which is 64)
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *  Notes:
 *  * version needs to be "1.000".
 *-----------------------------------------------------------------------*/
int
USERD_get_mf_version(char version_number[UDMFSNAME]) {

  strcpy(version_number, "1.000");
  return(Z_OK);
}

/*-----------------------------------------------------------------------
```

```
 * USERD_get_nargs
 *---------------------------------------------------------------------
 *
 *   Gets the number of arguments needed by the function.
 *
 *  (OUT) nArgs       = the number of arguments
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *---------------------------------------------------------------------*/
int
USERD_get_nargs(int *nArgs) {

  *nArgs = 2;
  return(Z_OK);
}

/*---------------------------------------------------------------------
 * USERD_get_meta_data
 *---------------------------------------------------------------------
 *
 *   Get the function descriptions, argument types, and return type.
 *
 *  (OUT) listDescription = description shown in general function column
 *  (OUT) funcDescription = description shown in feedback window
 *  (OUT) argTypes = data types of arguments passed into USERD_evaluate
 *  (OUT) returnType = data type returned by function USERD_evaluate
 *
 *  returns: Z_OK  if successful
 *           Z_ERR if not successful
 *
 *---------------------------------------------------------------------*/
int
USERD_get_meta_data(char listDescription[UDMFLNAME],
                    char funcDescription[UDMFLNAME],
                    int *argTypes, int *returnType) {

  strcpy(listDescription, "add2(part, scalar, scalar)");

  strcpy(funcDescription, "add2(any part(s), scalar, scalar)");

  argTypes[0] = UDMFSCL;
  argTypes[1] = UDMFSCL;

  *returnType = UDMFSCL;

  return(Z_OK);
}

/*---------------------------------------------------------------------
 * USERD_evaluate
 *---------------------------------------------------------------------
 *
 *   Evaluate the function.
 *
 *  (OUT) args = pointers to arguments
 *  (OUT) undefined = boolean; true if return value is undefined.
 *  (OUT) value = returned value
 *
 *  returns: Z_OK  if successful
```

```
 *           Z_ERR if not successful
 *
 *-------------------------------------------------------------------------*/
int
USERD_evaluate(void *args[], void *value, int *undefined) {

  float *result;
  float *arg1, *arg2;

  result = value;
  arg1 = args[0];
  arg2 = args[1];

  *result = *arg1 + *arg2;
  *undefined = 0;

  return(Z_OK);
}
```

5 EnSight Command Driver

Overview

This document provides information about a communication mechanism which can be used to drive EnSight from an external program using EnSight's command language. The logical steps involved in this process are:

1. Compile your external program with the enscmddriver_comm library.
2. Start EnSight and have it listen for the connection from the external program.
3. Start the external program and issue the connect command within that program.
4. Send commands to EnSight using the enscmddriver_sendmesg routine.
5. Shutdown Ensight.

More detail will now be provided for each of these steps.

Step 1:

Compile your external program with the enscmddriver.a library. This library is provided in the EnSight distribution, under the src/cmddriver directory. Directions for compiling are contained in the README file contained in that directory. Also provided therein is a sample external program (entitled enscmddriver.c) which is used to show how to compile, as well as for examples of how to utilize the following routines within your external driver:

enscmddriver_connect	To establish the connection with EnSight
enscmddriver_sendmesg	To send command language to EnSight
enscmddriver_query	To query information from EnSight (limited)
enscmddriver_disconnect	To disconnect and leave EnSight running (not commonly used)

Step 2:

Start EnSight and have it listen for the connection from the external program. Normally this will be done from your external program and will thus use batch mode to start EnSight.

In batch mode:

```
ensight100 -X -batch -externalcmds
```

While not the norm, it is possible to have EnSight start listening for the connection from an interactive session.

Interactively (from the command dialog in EnSight):

```
test: acceptcmddriver
```

Ensight will listen on Port 1104 for the connection from the external program. If a different port is desired, you can use the command line option "`-externalcmdport #`" when starting EnSight. Replace the `#` with a legitimate (>1024) port number. Then be sure to use the specified socket in the `enscmddriver_connect` call within your external program.

Step 3:

Once EnSight is listening, start the external program and issue the connect command within that program.

For the provided `enscmddriver` sample, this is done as follows:

```
enscmddriver HOSTNAME
```

Where, `HOSTNAME` is the name of the machine running EnSight. Note, the sample `enscmddriver` program calls the `enscmddriver_connect` routine to establish the connection.

Step 4:

Send commands to EnSight using the enscmddriver_sendmesg routine. The commands that you send to EnSight using this routine are the same commands that EnSight produces when users are manipulating a model with the EnSight Graphical User Interface. All of these commands are described in the Command Reference Manual within EnSight.

Note that the `enscmddriver_sendmesg` routine returns an `ok` or `ERROR` indicating its success or not.

It is possible to play entire command files that are accessible from the machine where the EnSight client is running. You can send a "play:" command to specify the name of the command file to use. Commands that are played using a file (play:) will execute faster than sending individual commands. The following is a command file (amiread.enc) that reads and colors the ami data set that is shipped with EnSight.

```
VERSION 7.52
data: binary_files_are big_endian
data: format case
data: path /usr/local/CEI/ensight100/data/ami
data: geometry ami.case
data: read
data_partbuild: begin
part: select_default
part: modify_begin
part: elt_representation not_loaded
part: modify_end
data_partbuild: data_type unstructured
data_partbuild: select_begin
 1
data_partbuild: select_end
data_partbuild: description
data_partbuild: create
part: select_default
part: modify_begin
part: elt_representation 3D_border_2D_full
part: modify_end
data_partbuild: data_type unstructured
data_partbuild: select_begin
 2
data_partbuild: select_end
data_partbuild: description
data_partbuild: create
data_partbuild: end
variables: activate pressure
part: select_all
part: modify_begin
part: colorby_palette pressure
part: modify_end
```

Your external program could send the command "play: amiread.enc" to Ensight using the `enscmddriver_sendmesg` routine. Ensight would play the command file, which would read in the model and color it by pressure, etc. It would then return and allow the external program to continue to issue other commands, such as would create images, produce VRML, create flipbook or keyframe animation sequences, etc.

There are also some commands that can be sent raw by the `enscmddriver_sendmesg` routine to bring up portions of EnSight's current GUI, so that user's can perform various actions. These "raw" commands are listed

in with the description of the enscmddriver_sendmesg rotuine. See GUI raw commands.

Additionally, the `enscmddriver_query` routine can be used to obtain some limited information back from EnSight. The current possibilities for this option will be described in the query section below.

Step 5:

Shutdown Ensight. If you did the normal, and started ensight in batch mode - you close the communication and get Ensight to stop by sending an `exit` command with the `enscmddriver_sendmesg` routine.

If you happen to be running EnSight interactively, rather than the normal batch mode, and you desire to close the connection, but leave EnSight running - you can use the `enscmddriver_disconnect` routine.

Example

Assuming that you were able to successfully compile our sample external program, `enscmddriver`, and that your machine name was “speedy”, you could do the following:

Start Ensight in batch mode (on your machine named “speedy”):

```
> ensight100 -X -batch -externalcmds &
```

Start the enscmddriver sample routine:

```
> enscmddriver speedy
```

Issue the following commands as prompted by the enscmddriver program:

```
What would you like to do?
play: amiread.enc
What would you like to do?
view: hidden_surface ON
What would you like to do?
savegeom: format vrml
What would you like to do?
savegeom: binary OFF
What would you like to do?
savegeom: save_geometric_entities /tmp/ami
What would you like to do?
exit
```

Which would read in the ami model using the amiread.enc command file, then turn shading on, then save a vrml file in /tmp. It would then close the communication and cause EnSight to exit.

You will of course be using your own external program, so the actual use of the enscmddriver_connect, and enscmddriver_sendmesg routines will be of interest to you. You can see them being used in the enscmddriver.c file. The routine arguments are described in detail in the Routine Descriptions section below.

5.1 Query Capability

The Ensight external command driver as first implemented with EnSight version 6.2.4, was purely a one-way interface. Namely, the external program could send command language to EnSight, but could not receive any type of information back (except for the error flag concerning success or failure of the command). Starting with EnSight 7.6, the capability to query EnSight for certain data has been added. While initially the scope of implemented queries is small, the implementation is general enough that future desirable queries should be easily added. Currently you can query for various transformation and viewport information.

The enscmddriver_query routine is driven by keywords. According to the keyword, the needed input parameters are defined, as well as the returned results.

Note: In the descriptions of the transformation matrices below, the components of a 4 x 4 matrix are:

```
| a11  a12  a13  a14 |
| a21  a22  a23  a24 |
| a31  a32  a33  a34 |
| a41  a42  a43  a44 |
```

Alphabetical List of Query Keywords:

ARROW_COUNT	PART_DISPLAY_ATTRIBUTES	TRANSFORMATION_COMPOSITE_MATRIX
ARROW_DISPLAY_ATTRIBUTES	PART_ELEMENT_PICKEDBYWINXY	TRANSFORMATION_LOOKAT_POSITION
ARROW_LABEL_TEXT		
ARROW_SELECTED_OBJECTS	PART_ELEMENT_PICKEDBYWORLDXYZ	TRANSFORMATION_LOOKFROM_POSITION
DIAL_COUNT	PART_NODE_PICKEDBYWINXY	TRANSFORMATION_PERANG
DIAL_DISPLAY_ATTRIBUTES	PART_NODE_PICKEDBYWORLDXYZ	TRANSFORMATION_PROJ_MATRIX
DIAL_SELECTED_OBJECTS	PART_OBJECTS	TRANSFORMATION_ROTATE_MATRIX
FLIPBOOK_INFORMATION	PART_PICKED	TRANSFORMATION_SCALE_MATRIX
FLIPBOOK_LOADED	PART_SELECTED_OBJECTS	TRANSFORMATION_TRANSLATE_MATRIX
FLIPBOOK_RUNNING	PLOT_COUNT	TRANSFORMATION_ZCLIP_LOCATIONS
FRAME_COUNT	PLOT_DISPLAY_ATTRIBUTES	VARIABLE_INFORMATION
FRAME_LOCATION	PLOT_PICKED	VARIABLE_OBJECTS
GAUGE_COUNT	QUERY_COUNT	VIEW_MODE
GAUGE_DISPLAY_ATTRIBUTES	QUERY_DISPLAY_ATTRIBUTES	VIEWPORT_COUNT
GAUGE_SELECTED_OBJECTS	QUERY_PICKED	VIEWPORT_DISPLAY_ATTRIBUTES
LEGEND_COUNT	QUERY_PROBE_ATTRIBUTES	VIEWPORT_LOCATION
	QUERY_PROBE_COUNT	
LEGEND_DISPLAY_ATTRIBUTES	QUERY_PROBE_OUTPUT	VIEWPORT_PICKED
LEGEND_SELECTED_OBJECTS	SHAPE_COUNT	VIEWPORT_SIZE
LINE_COUNT	SHAPE_DISPLAY_ATTRIBUTES	WINDOW_DEPTH_VALUES
LINE_DISPLAY_ATTRIBUTES	SHAPE_SELECTED_OBJECTS	WINDOW_MOUSECURRENT_INFO
LINE_SELECTED_OBJECTS	TEXT_COUNT	WINDOW_MOUSELASTPRESS_INFO
LOGO_COUNT	TEXT_DISPLAY_ATTRIBUTES	WINDOW_RGBA_VALUES
LOGO_DISPLAY_ATTRIBUTES	TEXT_DISPLAY_TEXT	WINDOW_SIZE
LOGO_SELECTED_OBJECTS	TEXT_SELECTED_OBJECTS	
MESSAGES	TRANSFORMATION_CENTER_OF	

Query Keyword Details

Count of 3D arrow annotations

```
Keyword:
  ARROW_COUNT

example command>  query ARROW_COUNT

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = number of 3D arrow annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

3D arrow annotation display attributes

```
Keyword:
  ARROW_DISPLAY_ATTRIBUTES

example command>  query ARROW_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = 3d arrow's id

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

3D arrow label text

```
Keyword:
  ARROW_LABEL_TEXT

example command>  query

Input:
  param_cnt      = 1
  param_array[0] = 3d arrow's id

Return Values:
  On Success -> (1)
    ret_str_cnt   = 1
    ret_str_array = 1D array containing the arrow label text

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected 3D arrows

```
Keyword:
  ARROW_SELECTED_OBJECTS

example command>  query ARROW_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected 3D arrows
    ret_int_array = 1D array of selected 3d arrow ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Count of dial annotations

```
Keyword:
  DIAL_COUNT

example command>  query DIAL_COUNT

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = number of dial annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Dial annotation display attributes

```
Keyword:
  DIAL_DISPLAY_ATTRIBUTES

example command>  query DIAL_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = dial's id

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected dials

```
Keyword:
  DIAL_SELECTED_OBJECTS

example command>  query DIAL_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected dials
    ret_int_array = array of selected dial ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Flipbook creation attributes

```
Keyword:
  FLIPBOOK_INFORMATION

example command>  query FLIPBOOK_INFORMATION

Input:

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Flipbook loaded or not?

```
Keyword:
  FLIPBOOK_LOADED

example command>  query FLIPBOOK_LOADED

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = 1(Yes), 0(No)

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Flipbook running or not?

```
Keyword:
  FLIPBOOK_RUNNING

example command>  query FLIPBOOK_RUNNING

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = 1(Yes), 0(No)

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of Frames

```
Keyword:
  FRAME_COUNT

example command>  query FRAME_COUNT

Input:
  param_cnt = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of frames

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Frame Location

```
Keyword:
  FRAME_LOCATION

example command>  query FRAME_LOCATION 1

Input:
  param_cnt      = 1
  param_array[0] = frame's id

Return Values:
  On Success -> (1)
    ret_float_cnt        = 12
    ret_float_array[0]   = x origin
    ret_float_array[1]   = y origin
    ret_float_array[2]   = z origin
    ret_float_array[3]   = x vector u
    ret_float_array[4]   = x vector v
    ret_float_array[5]   = x vector w
    ret_float_array[6]   = y vector u
    ret_float_array[7]   = y vector v
    ret_float_array[8]   = y vector w
    ret_float_array[9]   = z vector u
    ret_float_array[10]  = z vector v
    ret_float_array[11]  = z vector w


  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Count of gauge annotations

```
Keyword:
  GAUGE_COUNT

example command>  query GAUGE_COUNT

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt       = 1
    ret_int_array[0] = number of gauge annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Gauge annotation display attributes

```
Keyword:
  GAUGE_DISPLAY_ATTRIBUTES

example command>  query GAUGE_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = gauge's id

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected gauges

```
Keyword:
  GAUGE_SELECTED_OBJECTS

example command>  query GAUGE_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected gauges
    ret_int_array = array of selected gauge ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of legend annotations

```
Keyword:
  LEGEND_COUNT

example command>  query LEGEND_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of legend annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Legend's display attributes

```
Keyword:
  LEGEND_DISPLAY_ATTRIBUTES

example command>  query LEGEND_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = legend's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt       = number of attributes + 2
    ret_char_array[0] = legend description, format, and attribute commands with values
                        desc NULL format NULL command1 NULL command2 NULL ...
lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected legends

```
Keyword:
  LEGEND_SELECTED_OBJECTS

example command>  query LEGEND_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected legends
    ret_int_array = array of selected legend ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of line annotations

```
Keyword:
  LINE_COUNT

example command>  query LINE_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of line annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Lines's display attributes

```
Keyword:
  LINE_DISPLAY_ATTRIBUTES

example command>  query LINE_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = line's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes
    ret_char_array[0] = attribute commands with values
                        command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected lines

```
Keyword:
  LINE_SELECTED_OBJECTS

example command>  query LINE_SELECED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected lines
    ret_int_array = array of selected line ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of logo annotations

```
Keyword:
  LOGO_COUNT

example command>  query LOGO_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt         = 1
    ret_int_array[0]    = number of logo annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Logo's display attributes

```
Keyword:
  LOGO_DISPLAY_ATTRIBUTES

example command>  query LOGO_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = logo's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt       = number of attributes
    ret_char_array[0] = attribute commands with values
                          command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected logos

```
Keyword:
  LOGO_SELECTED_OBJECTS

example command>  query LOGO_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected logos
    ret_int_array = array of selected logo ids

  On Failure -> (-1)
    et_error_buf contains the error message string
```

Message Window contents

```
Keyword:
  MESSAGES

example command>  query MESSAGES

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_char_cnt      = 1
    ret_char_array[0] = message window contents

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Part's display attributes

```
Keyword:
  PART_DISPLAY_ATTRIBUTES

example command>  query PART_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = part's id as returned from "query PART_OBJECTS"

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes
    ret_char_array[n] = attribute commands with values
                        command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Element's id and X,Y,Z model location of pick point on the element

```
Keyword:
  PART_ELEMENT_PICKEDBYWINXY

example command>  query PART_ELEMENT_PICKEDBYWINXY x y

Input:
  param_cnt      = 2
  mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
  param_array[0] = x location
  param_array[1] = y location

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = element id
    ret_float_cnt      = 3
    ret_float_array[0] = X model location
    ret_float_array[1] = Y model location
    ret_float_array[2] = Z model location

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Element's id at model coordinate(X,Y,Z)

Keyword:
PART_ELEMENT_PICKEDBYWORLDXYZ

```
example command>  query PART_ELEMENT_PICKEDBYWORLDXYZ x y z

Input:
  param_cnt      = 3
  model location(X Y Z), so:
  param_array[0] = x location
  param_array[1] = y location
  param_array[2] = z location

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = element id

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Node's id and X,Y,Z model location of pick point on the node

Keyword:
PART_NODE_PICKEDBYWINXY

```
example command>  query PART_NODE_PICKEDBYWINXY x y

Input:
  param_cnt      = 2
  mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
  param_array[0] = x location
  param_array[1] = y location


Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = node id
    ret_float_cnt      = 3
    ret_float_array[0] = X model location
    ret_float_array[1] = Y model location
    ret_float_array[2] = Z model location

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Node's id at model coordinate(X,Y,Z)

```
Keyword:
  PART_NODE_PICKEDBYWORLDXYZ

example command>  query PART_NODE_PICKEDBYWORLDXYZ x y z

Input:
  param_cnt      = 3
  model location(X Y Z), so:
  param_array[0] = x location
  param_array[1] = y location
  param_array[2] = z location

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = node id

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Part general existence information

```
Keyword:
  PART_OBJECTS

example command>  query PART_OBJECTS

Input:
  param_array_cnt = 0

Return Values:
  On Success -> (1)
    ret_int_cnt      = number of parts + 1

    ret_int_array[0] = number of parts
    ret_int_array[1] = ID for part 1
    ret_int_array[2] = ID for part 2
    ret_int_array[3] = ID for part 3          .
          .
          .
          .
    ret_int_array[number of parts] = ID for last part

    ret_charstr_cnt  = number of parts
    ret_char_str     = name_of_part1 NULL name_of_part2 NULL ... name_of_lastpart NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Part's id and X,Y,Z model location of pick point on the part

Keyword:
PART_PICKED

example command> query PART_PICKED x y

Input:
param_cnt = 2
mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
param_array[0] = x location
param_array[1] = y location

Return Values:
On Success -> (1)
ret_int_cnt = 1
ret_int_array[0] = part id
ret_float_cnt = 3
ret_float_array[0] = X model location
ret_float_array[1] = Y model location
ret_float_array[2] = Z model location

On Failure -> (-1)
ret_error_buf contains the error message string

Part selection information

Keyword:
PART_SELECTED_OBJECTS

example command> query PART_SELECTED_OBJECTS

Input:
param_array_cnt = 0

Return Values:
On Success -> (1)
ret_int_cnt = number of selected parts + 1

ret_int_array[0] = number of selected parts
ret_int_array[1] = ID for part 1
ret_int_array[2] = ID for part 2
ret_int_array[3] = ID for part 3 .
.
.
ret_int_array[number of parts] = ID for last selected part

ret_charstr_cnt = number of selected parts
ret_char_str = name_of_part1 NULL name_of_part2 NULL ... name_of_lastpart NULL

On Failure -> (-1)
ret_error_buf contains the error message string

Number of plotters

```
Keyword:
  PLOT_COUNT

example command>  query PLOT_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of plotters

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Plotter's display attributes

```
Keyword:
  PLOT_DISPLAY_ATTRIBUTES

example command>  query PLOT_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = plotter's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes + 6
    ret_char_array[0] = plotter description NULL plotter title NULL
                        x axis title NULL y axis title NULL
                        x axis format string NULL y axis format string NULL
                        attribute commands with values separated by NULLs

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Plotter's id

```
Keyword:
  PLOT_PICKED

example command>  query PLOT_PICKED x y

Input:
  param_cnt      = 2
  mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
  param_array[0] = x location
  param_array[1] = y location

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = plotter id

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Number of queries

```
Keyword:
  QUERY_COUNT

example command>  query QUERY_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of queries

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Query's display attributes

```
Keyword:
  QUERY_DISPLAY_ATTRIBUTES

example command>  query QUERY_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = query's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes + 1
    ret_char_array[0] = query description NULL attribute commands
                        with values separated by NULLs

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Query item's id and value pair at pick point on curve

```
Keyword:
  QUERY_PICKED

example command>  query QUERY_PICKED x y

Input:
  param_cnt      = 2
  mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
  param_array[0] = x location
  param_array[1] = y location

Return Values:
  On Success -> (1)
    ret_int_cnt = 1
    ret_int_array[0] = query item id
    ret_float_cnt = 2
    ret_float_array[0] = X value
    ret_float_array[1] = Y value

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Query Probe's display attributes

```
Keyword:
  QUERY_PROBE_ATTRIBUTES

example command>  query QUERY_PROBE_ATTRIBUTES

Input:

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Count of probe queries

```
Keyword:
  QUERY_PROBE_COUNT

example command>  query QUERY_PROBE_COUNT

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = number of probe queries

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Query Probe's output

```
Keyword:
  QUERY_PROBE_OUTPUT

example command>  query QUERY_PROBE_OUTPUT

Input:
  param_cnt      = 2

Return Values:
  On Success -> (1)
    ret_char_cnt      = 1
    ret_char_array[0] = query probe output
    ret_char_cnt      = 0
    ret_char_array    = <No output>

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Count of shape annotations

```
Keyword:
  SHAPE_COUNT

example command>  query SHAPE_COUNT

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = number of shape annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Shape annotation display attributes

```
Keyword:
  SHAPE_DISPLAY_ATTRIBUTES

example command>  query SHAPE_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = shape's id

Return Values:
  On Success -> (1)
    ret_str_cnt   = total number of attributes
    ret_str_array = 1D array of command strings, each NULL terminated
                    command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected shapes

```
Keyword:
  SHAPE_SELECTED_OBJECTS

example command>  query SHAPE_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected shapes
    ret_int_array = array of selected shape ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of text annotations

```
Keyword:
  TEXT_COUNT

example command>  query TEXT_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of text annotations

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Text's display attributes

```
Keyword:
  TEXT_DISPLAY_ATTRIBUTES

example command>  query TEXT_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = text's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes
    ret_char_array[0] = attribute commands with values
                        command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Text's display text

```
Keyword:
  TEXT_DISPLAY_TEXT

example command>  query TEXT_DISPLAY_TEXT 1

Input:
  param_cnt      = 1
  param_array[0] = text's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = 1
    ret_char_array[0] = text annotation's text

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Selected text strings

```
Keyword:
  TEXT_SELECTED_OBJECTS

example command>  query TEXT_SELECTED_OBJECTS

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt   = total selected text strings
    ret_int_array = array of selected text string ids

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Center of Transformation

```
Keyword:
  TRANSFORMATION_CENTER_OF

example command>  query TRANSFORMATION_CENTER_OF 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt      = 3
    ret_float_array[0] = x coordinate of center of transformation
    ret_float_array[1] = y coordinate of center of transformation
    ret_float_array[2] = z coordinate of center of transformation

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Composite Transformation matrix - A combination of the look_at/look_from transform and the global transformation matrix.

```
Keyword:
  TRANSFORMATION_COMPOSITE_MATRIX

example command>  query TRANSFORMATION_COMPOSITE_MATRIX 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt       = 16            (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Lookat Position

Keyword:
TRANSFORMATION_LOOKAT_POSITION

```
example command>  query TRANSFORMATION_LOOKAT_POSITION 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt = 3
    ret_float_array[0] = x coordinate of lookat point
    ret_float_array[1] = y coordinate of lookat point
    ret_float_array[2] = z coordinate of lookat point

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Lookfrom Position

Keyword:
TRANSFORMATION_LOOKFROM_POSITION

```
example command>  query TRANSFORMATION_LOOKFROM_POSITION 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt      = 3
    ret_float_array[0] = x coordinate of lookfrom point
    ret_float_array[1] = y coordinate of lookfrom point
    ret_float_array[2] = z coordinate of lookfrom point

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Perspective angle

Keyword:
TRANSFORMATION_PERANG

```
example command>   query TRANSFORMATION_PERANG 0

Input:
  param_cnt      = 1
  param_array[0] = viewport number

Return Values:
  On Success -> (1)
    ret_float_cnt       = 1
    ret_float_array[0]  = perang

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Projection matrix

```
Keyword:
  TRANSFORMATION_PROJ_MATRIX

example command>   query TRANSFORMATION_PROJ_MATRIX 0

Input:
  param_cnt           = 1
  param_array[0]      = viewport number

Return Values:

  On Success -> (1)
    ret_float_cnt       = 16   (4 x 4 matrix)
    ret_float_array[0]  = a11
    ret_float_array[1]  = a12
    ret_float_array[2]  = a13
    ret_float_array[3]  = a14
    ret_float_array[4]  = a21
    ret_float_array[5]  = a22
    ret_float_array[6]  = a23
    ret_float_array[7]  = a24
    ret_float_array[8]  = a31
    ret_float_array[9]  = a32
    ret_float_array[10] = a33
    ret_float_array[11] = a34
    ret_float_array[12] = a41
    ret_float_array[13] = a42
    ret_float_array[14] = a43
    ret_float_array[15] = a44

  On Failure -> (-1)
    ret_error_buf contains the error message string

  where the matrix components are  | a11 a12 a13 a14 |
                                   | a21 a22 a23 a24 |
                                   | a31 a32 a33 a34 |
                                   | a41 a42 a43 a44 |
```

The Rotate Transformation matrix

```
Keyword:
  TRANSFORMATION_ROTATE_MATRIX

example command>  query TRANSFORMATION_ROTATE_MATRIX 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt       = 16            (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Scale Transformation matrix

```
Keyword:
  TRANSFORMATION_SCALE_MATRIX

example command>  query TRANSFORMATION_SCALE_MATRIX 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt       = 16            (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Translate Transformation matrix

```
Keyword:
  TRANSFORMATION_TRANSLATE_MATRIX

example command>  query TRANSFORMATION_TRANSLATE_MATRIX 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt       = 16            (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Zclip locations

```
Keyword:
  TRANSFORMATION_ZCLIP_LOCATIONS

example command>  query TRANSFORMATION_ZCLIP_LOCATIONS

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt      = 2
    ret_float_array[0] = near zplane z location
    ret_float_array[1] = far zplane z location

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Variable information - such as the active/inactive flag, current min and max values, expression for computed vars, etc.

```
Keyword:
  VARIABLE_INFORMATION

example command>  query VARIABLE_INFORMATION 1

Input:
  ret_param_cnt = 1
  ret_params[0] = variable number starting with 0 as returned
                  from "query VARIABLE_OBJECTS".

Return Values:
  On Success -> (1)
    ret_int_cnt         = 1
    ret_int_array[0]    = active flag (0 if inactive, 1 if active)

  If Computed
    ret_int_count       = 2 + number of parts used to compute it.
    ret_int_array[0]    = active flag (0 if inactive, 1 if active)
    ret_int_array[1]    = count of parts used to compute it
    ret_int_array[2]    = 1st part used to compute it.
    ret_int_array[3]    = 2nd part used to compute it.
          . . .
    ret_int_array[2+n] = nth part used to compute it.

  If Scalar:
    ret_float_cnt       = 2
    ret_float_array[0] = min value
    ret_float_array[1] = max value

  If Vector:
    ret_float_cnt       = 8
    ret_float_array[0] = x comp min value
    ret_float_array[1] = x comp max value
    ret_float_array[2] = y comp min value
    ret_float_array[3] = y comp max value
    ret_float_array[4] = z comp min value
    ret_float_array[5] = z compmax value
    ret_float_array[6] = magnitude min value
    ret_float_array[7] = magnitude max value

  If Computed:
    ret_charstr_cnt     = 1
    ret_char_str[0]     = command for calculator expression

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Variable general existence information

```
Keyword:
  VARIABLE_OBJECTS

example command>  query VARIABLE_OBJECTS

Input:
  param_array_cnt = 0

Return Values:
  On Success -> (1)
    ret_int_cnt      = number of vars + 1

    ret_int_array[1] = type for variable 1
    ret_int_array[2] = type for variable 2
    ret_int_array[3] = type for variable 3          .
          .
          .
    ret_int_array[number of vars] = type for last variable

    ret_int_array[1 + number of vars] = order for variable 1
    ret_int_array[2 + number of vars] = order for variable 2
    ret_int_array[3 + number of vars] = order for variable 3          .
          .
          .
    ret_int_array[2 * number of vars] = order for last variable

    ret_charstr_cnt  = number of variables
    ret_char_str     = name_of_var1 NULL name_of_var2 NULL ... name_of_lastvar NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string


  Map of variable types:
     0 = Scalar
     1 = Vector
     2 = Tensor
     3 = Scalar Complex
     4 = Vector Complex

  Map of variable orders:
     0 = Per Case (constant)
     1 = Per Elem
     2 = Per Node
```

Current view mode - Helps in determining which menu to bring up. For example a menu with part or plotter actions.

```
Keyword:
  VIEW_MODE

example command>  query VIEW_MODE

Input:

Return Values:
  On Success -> (1)
    ret_char_cnt      = 1
    ret_char_array    = a string value which is one of:
                        Part,Frame,Annot,Plot,VPort

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Number of viewports

```
Keyword:
  VIEWPORT_COUNT

example command>  query VIEWPORT_COUNT

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt        = 1
    ret_int_array[0]   = number of viewports

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Viewport's display attributes

```
Keyword:
  VIEWPORT_DISPLAY_ATTRIBUTES

example command>  query VIEWPORT_DISPLAY_ATTRIBUTES 1

Input:
  param_cnt      = 1
  param_array[0] = viewport's id (0-based)

Return Values:
  On Success -> (1)
    ret_char_cnt      = number of attributes
    ret_char_array[0] = attribute commands with values
                        command1 NULL command2 NULL ... lastcommand NULL

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Location of bottom left of Viewport - returned both as screen and as normalized coords

```
Keyword:
  VIEWPORT_LOCATION

example command>  query VIEWPORT_LOCATION 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt      = 2
    ret_float_array[0] = normalized window x coordinate of bottom left of viewport (0. to 1.)
    ret_float_array[1] = normalized window y coordinate of bottom left of viewport (0. to 1.)
    ret_int_cnt        = 3
    ret_int_array[0]   = screen x coordinate of bottom left of viewport
    ret_int_array[1]   = screen y coordinate of bottom left of viewport

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Viewport's id

```
Keyword:
  VIEWPORT_PICKED

example command>  query VIEWPORT_PICKED x y

Input:
  param_cnt      = 2
  mouse pointer location(X Y) as returned by WINDOW_MOUSELASTPRESS_INFO, so:
  param_array[0] = x location
  param_array[1] = y location

Return Values:
  On Success -> (1)
    ret_int_cnt      = 1
    ret_int_array[0] = viewport id

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Size of the Viewport, width and height - returned both as screen and as normalized values

```
Keyword:
  VIEWPORT_SIZE

example command>  query VIEWPORT_SIZE 1

Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)

Return Values:
  On Success -> (1)
    ret_float_cnt      = 2
    ret_float_array[0] = normalized window x size of viewport (0. to 1.)
    ret_float_array[1] = normalized window y size of viewport (0. to 1.)
    ret_int_cnt        = 3
    ret_int_array[0]   = screen x size of viewport
    ret_int_array[1]   = screen y size of viewport

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Window depth values

```
Keyword:
  WINDOW_DEPTH_VALUES

example command>  query WINDOW_DEPTH_VALUES

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_float_cnt      = xSize * ySize
    ret_float_array[n] = depth pixel interlaced values
                         if stereo, then the array is xSize * ySize * 2 and the
                         two stereo pairs are back to back

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

Mouse current location and button information

```
Keyword:
  WINDOW_MOUSECURRENT_INFO

example command>  query WINDOW_MOUSECURRENT_INFO

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt        = 7
    query_int_array[0] = x
    query_int_array[1] = y
    query_int_array[2] = left mouse button state
    query_int_array[3] = middle mouse button state
    query_int_array[4] = right mouse button state
    query_int_array[5] = Control key down(1=True,0=False)
    query_int_array[6] = window

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Mouse location and button information at last mouse button press

```
Keyword:
  WINDOW_MOUSELASTPRESS_INFO

example command>  query WINDOW_MOUSELASTPRESS_INFO

Input:

Return Values:
  On Success -> (1)
    ret_int_cnt        = 7
    query_int_array[0] = x
    query_int_array[1] = y
    query_int_array[2] = double click(1=True,0=False)
    query_int_array[3] = left mouse button state
    query_int_array[4] = middle mouse button state
    query_int_array[5] = right mouse button state
    query_int_array[6] = Control key down(1=True,0=False)

  On Failure -> (-1)
     ret_error_buf contains the error message string
```

Window RGBA values

```
Keyword:
  WINDOW_RGBA_VALUES

example command>  query WINDOW_RGBA_VALUES

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_char_cnt      = xSize * ySize * 4
    ret_char_array[n] = rgba pixel interlaced values
                        if stereo, then the array is xSize * ySize * 4 * 2 and the
                        two stereo pairs are back to back

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

```
Window size

Keyword:
  WINDOW_SIZE

example command>  query WINDOW_SIZE

Input:
  param_cnt      = 0

Return Values:
  On Success -> (1)
    ret_int_cnt         = 3
    ret_int_array[0]    = x size (pixels)
    ret_int_array[1]    = y size (pixels)
    ret_int_array[2]    = isStereo (boolean)

  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The supplied sample external routine (`enscmddriver.c`) contains an example of the use of this routine.

Please see the Routine Description section for an explanation of the other arguments to the enscmddriver_query routine.

5.2 Routine Descriptions

enscmddriver_connect

```
/*****************************************************************************
 *  Starts up connection to the EnSight client to drive it via commands.
 *  Parameters:
 *     host_toconnectto       - Character buffer containing hostname
 *                              where EnSight is running.
 *     sockport               - Port number to use for socket( > 1024).
 *     print_error            - if (1) will print errors to stderr when
 *                              they occur.
 *  Return Values:
 *    On Success              - Socket file descriptor to communicate with
 *                              EnSight, if success.
 *    On Failure
 *      ENS_SOCKRANGE         - Port number out of range. Must be > 1024.
 *      ENS_CONNECT           - Connection to EnSight failed.  EnSight must
 *                              be ready for the external command connection.
 *      ENS_HANDSHAKE         - The call to receive the handshake string
 *                              from the EnSight client failed.
 *      ENS_HOSTTOOLONG       - The hostname specified is too large.
 *****************************************************************************/
int
enscmddriver_connect(char *host_toconnectto,
                     int   sockport,
                     int   print_error)
```

enscmddriver_sendmesg

```
/*****************************************************************************
 *  This routine sends the EnSight client a command and waits for an ok (or ERROR).
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *     cmd                    - command string being sent
 *     print_error            - if (1) will print errors to stderr when
 *                              they occur.
 *  Return Values:
 *      1 - upon success
 *     -1 - upon failure
 *****************************************************************************/
int
enscmddriver_sendmesg(int   comm_socket,
                      char *cmd,
                      int   print_error)
```

Note:

The "cmd" argument can be basically any of the commands in the EnSight Command language. There are also some commands that can be sent raw by this routine to bring up portions of EnSight's current GUI, so that user's can perform various actions within those dialogs These current "raw" commands are listed below with the following form:

command_category: command <options delimited by "/">

Example: To open the quick interaction area for isosurfaces, you would send the following string to the `enscmddriver_sendmesg` routine:

```
isos: open_userinterface quick_interaction
```

The current possible commands are:

```
anim_flipbook: open_userinterface <quick_interaction>
anim_flipbook: close_userinterface <quick_interaction>

anim_keyframe: open_userinterface <quick_interaction>
anim_keyframe: close_userinterface <quick_interaction>

boundarylayer: open_userinterface <quick_interaction>
boundarylayer: close_userinterface <quick_interaction>

clip: open_userinterface <quick_interaction / detail_editor>
clip: close_userinterface <quick_interaction / detail_editor>

collab: open_userinterface <part / annotation / view / plot / viewport / frame /query_dataset / main_message>
collab: close_userinterface <part / annotation / view / plot / viewport / frame / query_dataset /main_message>

command: open_userinterface <string value>
command: close_userinterface <string value>

contour: open_userinterface <quick_interaction / detail_editor>
contour: close_userinterface <quick_interaction / detail_editor>

devsrf: open_userinterface <quick_interaction / detail_editor>
devsrf: close_userinterface <quick_interaction / detail_editor>

elevsurf: open_userinterface <quick_interaction / detail_editor>
elevsurf: close_userinterface <quick_interaction / detail_editor>

extrude: open_userinterface <quick_interaction / detail_editor>
extrude: close_userinterface <quick_interaction / detail_editor>

function: open_userinterface <detail_editor>
```

```
function: close_userinterface <detail_editor>

isos: open_userinterface <quick_interaction / detail_editor>
isos: close_userinterface <quick_interaction / detail_editor>

material: open_userinterface <quick_interaction / detail_editor>
material : close_userinterface <quick_interaction / detail_editor>

message_window: save <string value>
message_window: clear
message_window: open_userinterface
message_window: close_userinterface
message_window: display_information <string value>
message_window: display_warning <string value>
message_window: display_error <string value>
message_window: display_errorpopup <string value>

model: open_userinterface <quick_interaction / detail_editor>
model: close_userinterface <quick_interaction / detail_editor>

part: open_userinterface <quick_interaction_color / quick_interaction_displace / part_viewer>
part: close_userinterface <quick_interaction_color / quick_interaction_displace / part_viewer>

pointpart: open_userinterface <quick_interaction / detail_editor>
pointpart: close_userinterface <quick_interaction / detail_editor>

profile: open_userinterface <quick_interaction / detail_editor>
profile: close_userinterface <quick_interaction / detail_editor>

ptrace: open_userinterface <quick_interaction / detail_editor>
ptrace: close_userinterface <quick_interaction / detail_editor>

query_ent_var: open_userinterface <quick_interaction / quick_interaction_query / quick_interaction_plot>
query_ent_var: close_userinterface <quick_interaction / quick_interaction_query /quick_interaction_plot>

query_interact: open_userinterface <quick_interaction>
query_interact: close_userinterface <quick_interaction>

sepattach: open_userinterface <quick_interaction / detail_editor>
sepattach: close_userinterface <quick_interaction / detail_editor>

shock: open_userinterface <quick_interaction / detail_editor>
shock: close_userinterface <quick_interaction / detail_editor>

solution_time: open_userinterface <quick_interaction>
solution_time: close_userinterface <quick_interaction>

subset: open_userinterface <quick_interaction / detail_editor>
subset: close_userinterface <quick_interaction / detail_editor>

tensor: open_userinterface <quick_interaction / detail_editor>
tensor: close_userinterface <quick_interaction / detail_editor>

variables: open_userinterface <quick_interaction / detail_editor>
variables: close_userinterface <quick_interaction / detail_editor>

vctarrow: open_userinterface <quick_interaction / detail_editor>
vctarrow: close_userinterface <quick_interaction / detail_editor>

vortexcore: open_userinterface <quick_interaction / detail_editor>
vortexcore: close_userinterface <quick_interaction / detail_editor>
```

enscmddriver_query

```
/*****************************************************************************
 *  This routine sends the EnSight client a query command and waits for the results.
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *     query_keyword          - Query keyword
 *     param_array_cnt        - Count of parameters in array below.
 *     param_array            - Floating point array containing any parameters
 *                              for the query operation.  The count above helps
 *                              to clarify any changes that might be made to
 *                              a particular query in the future.  This will
 *                              help to allow forward/backward compatibility
 *                              and prevent users from always having to use
 *                              the latest library.
 *
 *     ***NOTE: the next 6 need to be passed in by address(ex. &ret_int_cnt)
 *              because return values will be placed in the ...cnt variables
 *              and space will be allocated for the others and return
 *              information will be placed in this space.
 *     ret_charstr_cnt        - Count of strings concatenated into string return
 *     ret_char_str           - String(s) returned from query and separated
 *                              by NULLs.  When the user finishes with the
 *                              information they must use free() to deallocate.
 *     ret_int_cnt            - Count of integers in return int array.
 *     ret_int_array          - Array of integer return values.   When the user
 *                              finishes with the information they must use
 *                              free() to deallocate.
 *     ret_float_cnt          - Count of floats in return float array.
 *     ret_float_array        - Array of float return values.   When the user
 *                              finishes with the information they must use
 *                              free() to deallocate.
 *
 *     ret_error_buf          - Buffer for error return string.  This buffer
 *                              should be preallocated to 500 characters by
 *                              the caller.  It will contain a NULL terminated
 *                              error string when the return value is -1.
 *
 *  Return Values:
 *    On Success              - (1)
 *    On Failure              - (-1)  (See error_buffer above)
 *
 *****************************************************************************/
int
enscmddriver_query(int     comm_socket,
                   char   *query_keyword,
                 int     param_array_cnt,
                   float  *param_array,
                   int    *ret_charstr_cnt,
                   char  **ret_char_str,
                   int    *ret_int_cnt,
                   int   **ret_int_array,
                   int    *ret_float_cnt,
                   float **ret_float_array,
                   char    ret_error_buf[500])
```

enscmddriver_disconnect

```
/*****************************************************************************
 * This routine cleans up the connection to EnSight.  This must
 * be done before you exit, especially if your application is dieing
 * because it received a signal.  If the socket is not closed properly
 * your port may become hung and you won't be able to use it until
 * it is cleared out by a reboot of your system or some other event.
 *
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *
 *   Return Values:
 *      None
 *****************************************************************************/
void
enscmddriver_disconnect(int comm_socket)
```

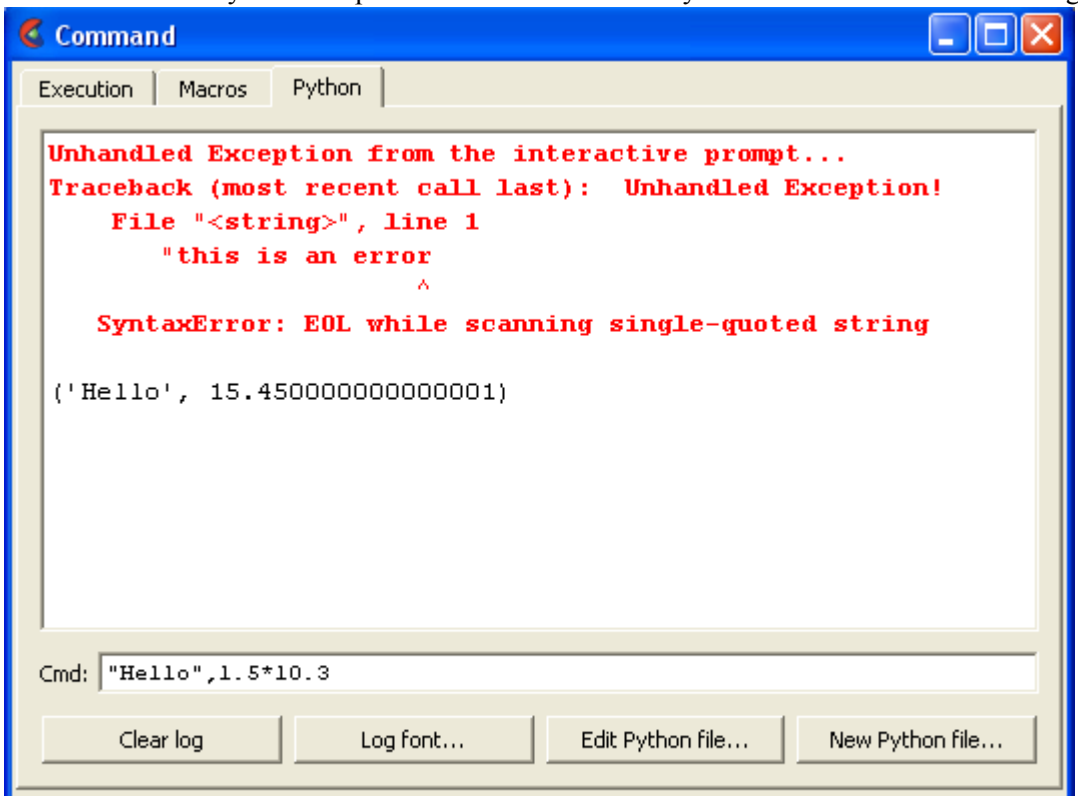
6 EnSight Python Interpreter

Overview

EnSight includes a built-in interpreter for the Python programming language (www.python.org). The system allows Python code to be executed within the EnSight program, not unlike the command language allows. Python is a more fully featured programming language with formal flow control, classes and complex variable types. It is also intrinsically extensible. EnVe is an example of a Python extension. The popularity of the Python language means that there are a large number of available extensions (e.g. xml, SQL, COM, etc). The Python built into EnSight includes the core classes and libraries as well as the EnSight, EnVe, numpy and PyQt (Python interface to the Qt GUI library) modules. The PyQt module allows Python code running inside of EnSight to create cross-platform custom GUIs that can interact with EnSight. The EnSight core itself implements extensions (command language, right mouse button menus, user-defined tools and user-defined GUIs) in Python.

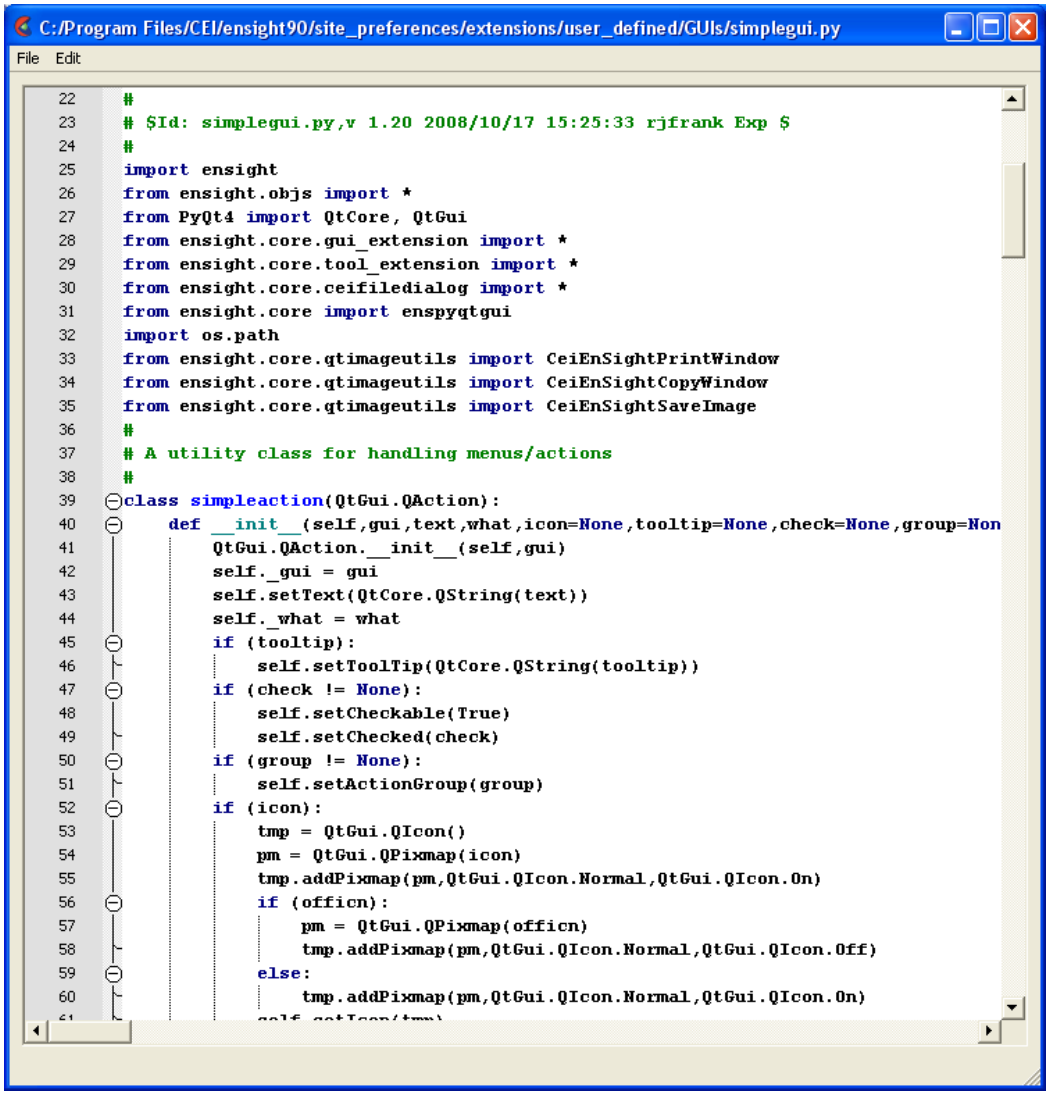
EnSight uses Python, Qt and PyQt. The Python interpreter uses version 2.5.1 and generally includes the modules built from the standard Python source code distribution. Qt is version 4 with corresponding updates to QScintilla and PyQt. See the Qt and PyQt websites for more details on this change.

The core interface to the Python interpreter is accessed via the "Python" tab in the command dialog GUI



The 'Cmd:' edit field allows the user to interactively type in Python commands. The output of those commands is captured in the pane above the prompt. Normal output from Python is in black text, while error output is displayed as red text. Simple, one-line commands can be entered and executed when the user presses 'enter'. The command prompt allows for command recall as well. The up and down arrow keys walk though the most recently entered commands, facilitating rapid editing and re-issuing of commands. A button is provided to clear the current log text at any time. The ‘Log font...’ button allows the user to change the font used in the dialog as well as the Python source code editor.

EnSight provides a built-in editor for Python code that includes Python aware syntax highlighting. Buttons are provided to create a new Python script file or edit an existing one. The editor window looks like this:



The line numbers are down the left side and a column is provided to allow the user to hide/show blocks of text. This also makes it easier to see how the block indented structure of Python denotes scope. The menu options allow for many options. In addition to basic file I/O and cut/copy/paste editing, there is a find/replace dialog set and options to indent/unindent and block comment. A menu allows the user to adjust the font used for display. This selection is stored in the user's preferences.

The editor provides a simple auto-completion mechanism that can be enabled from the 'Edit' menu or by typing the 'ctrl-tab' key. When enabled, if the '.' character is typed, the text to the left of the '.' is scanned for a Python module name match and items in that module are displayed for easy selection. For example, entering 'ensight.' brings up a list of objects in the Python ensight module. The can be very helpful in getting the proper syntax, especially of seldom used commands. Note that the system will work for the EnSight core Python modules as well as any module that has been imported into the global Python interpreter namespace.

There is a built-in mechanism to convert code in command language into Python. To do this, you first paste the command langauge into the Python editor (lines of command language can be selected and copied using the right mouse button menu in the "Execution" tab). Next, select the text in the editor and used the Edit menu

options for “Convert selection to sendmesg()” or “Convert selection to native Python”. In general, the ‘native Python’ conversion results in much more readable Python code that is far easier to edit than the sendmesg() option. The native option should be used for all but legacy development.

The file menu provides two items to execute the current file text in the EnSight Python interpreter. The “Run script” item causes the file contents to be executed in the global namespace (e.g. like the execfile() function). The “Import script as module” item first saves the current file to disk and then executes a Python ‘import’ operation on the file, which executes in a private namespace. Both will check the syntax of the current file and allow for rapid prototyping.

General Python use in EnSight

EnSight will accept Python scripts in most situations where it is expecting a command script. For example, in the 'Load:' prompt in the command dialog, as the name of a keyboard or HUM macro and on the command line (-p), the user may specify the name of a Python script to be executed. This would allow things like GUIs to be popped up when keys are pressed, etc. The command language 'play: filename' command will accept a Python filename, allowing the execution of Python code from any command file. EnSight differentiates between command language files and Python files based on filename suffix. Python files are assumed to end in '.py' or '.pyc'. All other files are assumed to contain EnSight command language instructions. Unlike the command language, EnSight Python code is not journaled during execution by default, thus Python commands will not show up in any saved EnSight command stream unless special flags are set.

Simple Python commands can also be embedded directly in EnSight command language. If a line of command language begins with ‘ensight.’, it is assumed to be Python code and is passed directly to the Python interpreter. This means that the “native” API, discussed later in this section, can be used directly in .enc files. Also, the command language command: “ext: python” has been added to command language. This command passes the remainer of the command line to the Python interpreter. Thus in EnSight the following is a legal .enc file:

```
VERSION 9.2
ensight.view_transf.rotate(0.0,0.0,0.1)
ext: python print "Hello from Python!"
```

Note that the printed string will appear in the Python tab in the command dialog.

Python and the EnSight command line

There are a number of situations where it would be helpful for the Python interpreter in EnSight to have access to the EnSight command line. This is done through the ‘sys.argv’ interface in Python. The EnSight client application supports the command line options ‘-pyargv’ and ‘-endpyargv’. These options serve to mark a section of the command line used to launch EnSight to be passed to the EnSight Python interpreter. For example:

```
ensight100_client -c -pyargv example -another_example -endpyargv -X
```

Will cause EnSight to be launched with the ‘-c’ and ‘-X’ options and the value of sys.argv to be set to [“ensight”,”example”,”-another_example”] in Python. The use of -endpyargv is optional and if missing, will cause all the arguments followin ‘-pyargv’ to be passed to sys.argv. Note: EnSight will prepend the string “ensight” to the start of sys.argv. If the raw command line is desired, it is available as the ‘ensight.argv’ list. In the above example, the output would be something like: ['C:\\Program Files\\CEI\\ensight100\\machines\\win64\\ens100cl.exe', '-hide_console', '-c', ‘-pyargv’, ’example’, ’-another_example’, ’-endpyargv’, ‘-X’, '-iwd', '-homecwd']

Limitations of the EnSight Python Interface

Python is a complex and broad-ranging programming language. There are a few features of the language that can cause problems if called from within EnSight. The features should not be used by Python code running inside of EnSight.

Re-entrant interfaces to the command language are not allowed. For example, a Python script may use ensight.sendmesg() to 'play:' a command language file. If that file in turn tries to execute a Python script, EnSight will fail. The reverse is also true. If a command language script calls a Python script which then calls a command language script, EnSight will fail. In general, avoid nesting 'play:' commands that change interpreters.

Python supports threads and you can use these, except there are two problems. Python threads require the interpreter be running at all times to execute. In EnSight, the interpreter is dormant unless Python code is being executed, so the threads may not always be executed. Second, threads that explicitly or implicitly modify GUI elements can cause issues. In EnSight, it is critical that only the main thread of execution make GUI changes. Note that even simple things like 'print' in EnSight Python cause GUI elements to change (the output is logged to a Qt widget). The best advice is not to use Python threads in EnSight.

The PyQt module provides a socket interface. This interface is based on asynchronous socket calls. While the interface is quite nice, it has the side effect of making all the other socket calls in EnSight under Windows asynchronous. This will cause EnSight's socket communication library to fail. If you need a socket connection in Python, use the provided Python module instead of the PyQt module.

EnSight balances the X11 Motif widgets with those provided by PyQt, allowing both to exist and co-operate. One exception to this rule is the issue of modal widgets. The current event handling system cannot properly handle the case of both widget systems having a modal widget. Thus, this case must be avoided. The most common situation occurs when a modal PyQt widget is active and a callback function on that widget uses ensight.sendmesg() and the resulting command causes EnSight to pop up a modal X11 dialog. This will cause EnSight to hang. The work-around is to avoid calling sendmesg() when there is a modal PyQt widget (note that Qt popup menus are modal widgets). Always allow the modal operation to complete before making the sendmesg() calls.

In order to provide a more seamless integration between the EnSight core windows and those generated via the Python API, special consideration must be given to the parents of top-level widgets. Two methods, ensight.qtparent() and ensight.reparent(), provide this functionality. Proper use of these methods is to provide ensight.qtparent() as the parent of any top-level Qt widget and to call ensight.reparent() in the widget __init__() method. An Example would be:

```
class example(QWidget):
  def __init__(self, parent=None, flags=Qt.WType_TopLevel):
    QWidget.__init__(self, parent, flags)
    ensight.reparent(self)
    ...
temp = example(ensight.qtparent())
temp.show()
```

The resulting top-level widget will interact properly with the EnSight GUI windows.

6.1 Python EnSight module interface

Key to interacting with EnSight from within Python is the 'ensight' module. This module is pre-loaded into the environment and provides a number of methods that can be used to communicate directly with EnSight. There are a large number of examples that use this interface included in the user-defined EnSight extensions. These can be found in the $CEI_HOME/ensight100/site_preferences/extensions/user_defined directory tree.

Interpreter startup

When the EnSight Python interpreter is initialized it performs the following operations:

1. Set up PYTHON_HOME/PATH.
2. Create the ‘ensight’ module.
3. Create the cmdlanguage methods in the ‘ensight’ module for the native API.
4. Import the sys, sip and imp modules.
5. Import the PyQt4.Gui module.
6. Add $CEI_HOME/ensight100/site_preferences/extensions to sys.path.
7. If it exists, add the EnSight Defaults Directory path plus the `extensions` subdirectory to sys.path. The EnSight Defaults Directory path is %HOMEDRIVE%%HOMEPATH%\(username)\.ensight100 commonly C:\Users\username\.ensight100 on Vista and Win7, C:\Documents and Settings\yourusername\.ensight100 on older Windows, and ~/.ensight100 on Linux, and ~/Library/Application Support/ensight100 on the Mac.
8. Import the ensight.core module. This has the side effect of adding the names specified by the CEI_PYTHONPATH environmental variable (“:” separated under Unix and “;” separated under Windows) to sys.path.
9. Execute ensight.core.load_extensions(). This walks the extension tree, loads any user defined extensions and reads in any Qt4 language files (*.qm) it finds.

Ensight Module Methods

`ensight.sendmesg(cmd|(cmdlist)[,record=r] [,display=d] [,exception=e] [,version=v] [,noexec=n])`	

	This method executes one or more command language commands. The function can be passed a single string or a tuple of strings. In the latter case, all the strings in the tuple will be executed. For example:
	`ensight.sendmesg(("shell: echo A", "shell: echo B"))`
	executes two command language commands.
	The sendmesg() function provides mechanisms to control the display and recording of the individual command language commands it executes. The 'record=' and 'display=' keyword arguments control this. By default, commands are not recorded in the session command log. If the record keyword is set to '1', the commands will be recorded in the session log. For example:
	`ensight.sendmesg("shell: echo hello",record=1)`
	will execute the command and cause it to be recorded. If the display keyword is set to '1', the commands will be echoed as text as they are executed to the Python text log in the command dialog Python tab. Similarly, the 'noexec=' keyword controls if the command being issued should actually be executed. Setting noexec to 1 supresses actual execution of the command, but still allows for the command to be displayed or recorded as selected by other keywords.
	This method returns a 0 if no error and a -1 if error. For example we can purposely enter in the correct rotate command and an incorrect rotate command and show the return values:
	`a = ensight.sendmesg("view_transf: rotate -1 1 0")`
	`print a`
	`0`
	`a = ensight.sendmesg("view_transf: rotERRORate -1 1 0")`
	`print a`
	`-1`
	If the exception keyword is set to '1', and the command language string resulted in an error, Python will raise a Python exception instead of returning -1, which can be trapped by normal Python exception handling mechanisms.
	If the version keyword is set, the version of the command language will be set to that version for the duration of this single command. The default is to use the version set by sendmesgoptions(). If the keyword is set to 0, the current version of EnSight is used.
	Explicitly setting any keywords will override the defaults set by the ensight.sendmesgoptions() method.
	Note, you can send the commands that open a part of the current EnSight GUI with this method. See GUI raw commands

ensight.sendmesgoptions([record=r] [,display=d] [,exception=e] [,version=v] [,noexec=n])	

	This method allows the caller to set the default values for the ‘exception’, 'record', 'display', ‘version’ and ‘noexec’ keywords in the ensight.sendmesg() method. By default, the keywords except ‘version’ are set to 0. If they are set to 1 by this method, subsequent sendmesg() calls without the keywords specified will use the values set by this method. For example:
	`ensight.sendmesgoptions(record=1,display=1)`
	`ensight.sendmesg("shell: echo A")`
	will result in the shell: command being both logged and printed. The commands:
	`ensight.sendmesgoptions(record=1,display=1)`
	`ensight.sendmesg("shell: echo A",record=0)`
	would result in the shell: command being printed, but not recorded. The noexec keyword can be set to 1 to allow the record and display operations to run as selected, but actual execution of the line of command language is supressed.
	By default, ‘version’ is set to the current EnSight command language version number. The keyword can be set to any valid floating point version number. If it is set to 0, the version will be reset to the version of EnSight running (its initial state).

(value,type,scope) = ensight.ensvariable(varname)	
	This method will query EnSight command language variable values and returns a tuple containing the value, its type and the scope it was found in (similar to the commnad language ‘$’ functionality, allowing access to variables defined directly in command language or computed constants). The function returns None if the variable cannot be found.The TYPE of the variable will be 0 if the value is an integer, 1 if the value is a float and 2 if the value is a string.The SCOPE is an integer that specifies where the variable was found. If the variable is a computed constant, the scope is -1; if it is a varaible defined by the command language, the scope will be 0 if global and greater than zero if the value is local to a file or loop. See the Command Language Manual, under “Scripting Constructs” for details.

ensight.play(filename)	
	This method causes the command language file (.enc) specified by FILENAME to be executed (similar to the ‘play: filename’ command). If the name of a Python (.py) file is specified, the file will be read into memory and executed with the Python ‘exec()’ command.

ensight.reparent(QWidget_object)	
	This method is useful on X11 platforms and harmless under Windows. It causes any PyQt widgets one might create in the EnSight Python interpreter to properly layer with the EnSight windows (keeping them above the main EnSight window). Note, calling this function with the wrong argument type can cause EnSight to crash.

```
(err, value, ...) = ensight.query(param[,which])
```

This method allows the caller to query various attributes in EnSight. The returned value is always a list that starts with an error code. The list will include one or more returned values (specific to the param). See ?$paratext>? on page 4.

Valid "param" values include the following. Note that an "*" means the "which" argument is required:

```
ensight.FRAME_COUNT
ensight.FRAME_LOCATION*
ensight.LEGEND_COUNT
ensight.LEGEND_DISPLAY_ATTRIBUTES *
ensight.LEGEND_SELECTED_OBJECTS
ensight.LINE_COUNT
ensight.LINE_DISPLAY_ATTRIBUTES *
ensight.LINE_SELECTED_OBJECTS
ensight.LOGO_COUNT
ensight.LOGO_DISPLAY_ATTRIBUTES *
ensight.LOGO_SELECTED_OBJECTS
ensight.DIAL_COUNT
ensight.DIAL_DISPLAY_ATTRIBUTES *
ensight.DIAL_SELECTED_OBJECTS
ensight.FLIPBOOK_INFORMATION
ensight.FLIPBOOK_LOADED
ensight.FLIPBOOK_RUNNING
ensight.GAUGE_COUNT
ensight.GAUGE_DISPLAY_ATTRIBUTES *
ensight.GAUGE_SELECTED_OBJECTS
ensight.SHAPE_COUNT
ensight.SHAPE_DISPLAY_ATTRIBUTES *
ensight.SHAPE_SELECTED_OBJECTS
ensight.ARROW_COUNT
ensight.ARROW_DISPLAY_ATTRIBUTES *
ensight.ARROW_SELECTED_OBJECTS
ensight.ARROW_LABEL_TEXT
ensight.MESSAGES
ensight.PART_DISPLAY_ATTRIBUTES *
ensight.PART_ELEMENT_PICKEDBYWINXY * *
ensight.PART_ELEMENT_PICKEDBYWORLDXYZ * * * *
ensight.PART_NODE_PICKEDBYWINXY * *
ensight.PART_NODE_PICKEDBYWORLDXYZ * * * *
ensight.PART_OBJECTS
ensight.PART_PICKED * *
ensight.PART_SELECTED_OBJECTS
ensight.PLOT_COUNT
ensight.PART_DISPLAY_ATTRIBUTES *
ensight.PLOT_PICKED * *
ensight.QUERY_COUNT
ensight.QUERY_DISPLAY_ATTRIBUTES
ensight.QUERY_PICKED * *
ensight.QUERY_PROBE_DATA
ensight.QUERY_PROBE_OUTPUT *
ensight.QUERY_PROBE_ATTRIBUTES
ensight.TEXT_COUNT
ensight.TEXT_DISPLAY_TEXT *
ensight.TEXT_DISPLAY_ATTRIBUTES *
ensight.TEXT_SELECTED_OBJECTS
```

ensight.TRANSFORMATION_PERANG * ensight.TRANSFORMATION_PROJ_MATRIX * ensight.TRANSFORMATION_CENTER_OF * ensight.TRANSFORMATION_COMPOSITE_MATRIX * ensight.TRANSFORMATION_LOOKAT_POSITION * ensight.TRANSFORMATION_LOOKFROM_POSITION * ensight.TRANSFORMATION_ROTATE_MATRIX * ensight.TRANSFORMATION_SCALE_MATRIX * ensight.TRANSFORMATION_TRANSLATE_MATRIX * ensight.TRANSFORMATION_ZCLIP_LOCATIONS * ensight.VARIABLE_OBJECTS ensight.VARIABLE_INFORMATION * ensight.VIEWPORT_COUNT ensight.VIEWPORT_DISPLAY_ATTRIBUTES * ensight.VIEWPORT_LOCATION * ensight.VIEWPORT_SIZE * ensight.WINDOW_SIZE ensight.WINDOW_DEPTH_VALUES ensight.WINDOW_RGBA_VALUES †
† The ensight.WINDOW_RGBA_VALUES is a special case. The second argument value picks the format of the returned data. A 0 returns the image as a list of integers, 1 returns the image as an ASCII string (which happens to be a valid PPM file) and 2 returns the image as an enve image object

The following query param() options are specific to the Python interface and are not supported by the command driver interface:

ensight.TEXTURE_COUNT	
	Returns the number of textures EnSight supports.
ensight.TEXTURE_IMAGE *	
	Returns the texture selected by param1 as an EnVe image object (see the EnVe module description of the image object)
ensight.TEXTURE_BORDER_COLOR *	
	Returns the texture border color [R,G,B,A] selected by param1.
ensight.VARIABLE_PALETTE * [*]	
	Returns the current palette for the variable index (see ensight.VARIABLE_OBJECTS) selected by param1. If the variable is a vector, param2 selects the palette for the magnitude (0), x (1), y (2) or z (3) components (it defaults to 0). Each palette entry is four values in the floating point array. The first is the value and the next three are the R, G, B color for that value.
ensight.CMDLANG_VERSION	

	Returns the current EnSight command language version number being used. The values is returned as a string.
`ensight.VARIABLE_HISTOGRAM *`	
	Returns the min, max and current histogram for the variable index (see ensight.VARIABLE_OBJECTS) selected by param1. For a scalar variable, 102 floating point values are returned. The first two are the variable min and max respectively. The subsequent 100 values are the counts for 100 bins between the min and max values. For vector variables, four times the number of values are returned. Values for the magnitude and x, y, and z components are included.
`ensight.QUERY_DATA *`	

	For the query selected by param1 (see `ensight.QUERY_COUNT`), this values returns the actual point data for that query. The routine returns integer, string and float values. The integers start with the number of columns. These will be 2 or 5 depending on the type of query (e.g. over time or over distance). The remaining integers define the number of point that go into each segment. The (2 or 5) strings are the labels for the columns. The floating point values will be either 2 or 5 per point and there will be as many points as the sum of the integers following the number of columns.
	
	Example:
	
	`num_queries = ensight.query(ensight.QUERY_COUNT)`
	`print "The number of queries is" , num_queries[1]`
	`for query_num in range(num_queries[1][0]):`
	`  query_vals = ensight.query(ensight.QUERY_DATA,query_num)`
	`  print "query vals, query num", query_num,"=", query_vals`

`ensight.QUERY_PROBE_DATA`	

	Returns the current interactive probe query values. The output is a list, one item per query value. Each item is a list of the form:
	
	`[[variables], [xyz], [ijk], ID, type]`
	
	The first item in the output list contains text headers for all the columns. The ‘variables’ list contains one item for each selected variable. The ‘xyz’ list contains a list of three floats, the coordinates of the query. The ‘ijk’ list (which can be null) contains a list of three ints, the ijk values of the query (if any). If the query has an ID value, it will be returned ad ‘ID’ (or the value will be null). The ‘type’ of the query is an ascii string of the source of the query. Note: ‘Undefined’ values will be returned as Python None objects.
	
	Example for a simple query with two probes and two variables:
	
	`print ensight.query(ensight.QUERY_PROBE_DATA)`
	`[`
	`  [[‘temperature’, ‘velocity’], [‘X’,’Y’,’Z’], [‘I’,’J’,’K’], ‘ID’, ‘Type’],`
	`  [[0.9, 0.9], [1.5, -2.0, 0.9], [None, None, None], None, ‘Surface Pick’],`
	`  [[0.9, 0.9], [-0.8, -2.0, 0.9], [None, None, None], None, ‘Surface Pick’]`
	`]`

`ensight.DATASET`

Returns information about the current dataset. The output may be “None” if the current case is not loaded of a list of the form:

```
[[filedata],coord_sys,coord_type,full_flag,[[minx,miny,minz],[maxx,maxy,maxz]], total_nodes, total_elements, [elementlist]]
```

The FILEDATA list describes the various files opened (as specified by the actual reader). The information includes the filename, it’s size, it’s date and any number of description strings. The list has the form:

[[filename,file_size,file_date,[“desc1”,”desc2”,...]]

COORD_SYS specifies the file’s coordinate system and can be: “Cylindrical”, “Rectangular”, “Spherical” or “Unknown”. COORD_TYPE specifies the file’s temporal scheme and can be: “Static”, “Changing Connectivity” or “Changing Coordinate”.

The FULL_FLAG is 1 if the min/max arrays are for the entire dataset. Otherwise the numbers are just for the loaded parts. The subsequent arrays specify the coordinate min/max values for x, y and z.

TOTAL_NODES and TOTAL_ELEMENTS specify the total number of nodes and elements in the dataset respectively. The ELEMENTLIST list describes the number of elements in the dataset of each element type. The list has the form:

[[“type1”,num_elements1],[“type2”,num_elements2],...]

`ensight.TIMEVALS`

This query returns a dictionary of the current EnSight timeline. The dictionary keys include:

timecurrent - the current time value
stepcurrent - the current timestep (note: this is a float as it could be interpolated)
timelimits - list of the minimum and maximum time values
steplimits - list of the minimum and maximum time steps (integers)
timevalues - list of [timestep, timevalues] lists for every simulation time

`ensight.FULLSCREEN`

This query returns zero if EnSight is not in fullscreen mode or non-zero if it is in fullscreen mode.

`ensight.PREFERENCESPATH`

This query returns a string that is the directory name where EnSight is reading and writing preference data to.

`ensight.SENDMESG_RECORD,SENDMESG_RAISE,SENDMESG_NOEXEC,SENDMESG_DISPLAY`

	These four queries return the current value of the ‘record’, ‘exception’, ‘noexec’ or ‘display’ named agruments respectively to the ensight.sendmesgoptions() method.

ensight.GLCONFIG	
	This query returns a dictionary of OpenGL related information. The dictionary keys include (on most platforms): vendor - the OpenGL vendor string renderer - the OpenGL renderer string version - the OpenGL version string stencilbuffer - a non-zero integer if a stencil buffer is being used doublebuffer - a non-zero integer if depth buffering is being used occlusiontest - True if the OpenGL occlusion test is being used wirefamemode - True if wireframe mode is done with lines (instead of polygons) displaylist - True if OpenGL display lists are being used normpervertex - True if EnSight is generating a normal per vertex vertexcount - the number of vertices in a geometry block palettemode - “Textures” or “RGB” extensions - the OpenGL extension string

ensight.SYSINFO	
	This query returns a dictionary of system resource related information. The dictionary keys include (on most platforms): numprocs - number of processor (cores) in the system totalmem - amount of physical RAM in the system (in kilobytes) freemem - amount of free RAM in the system (in kilobytes) memuse - the amount of RAM being used by the EnSight process (in kilobytes) pointersize - the size of a pointer in bits on the platform (32 or 64)

ensight.DR_INFO	
	This query returns a dictionary of system resource related information. The dictionary keys include (on most platforms): parallelrendering - a non-zero integer if the system is using parallel rendering masterclient - True if the current node is the DR master client compositing - True if image-based parallel compositing is being used detached - True if DR detached displays are being used rank - the rank of the current EnSight client node in DR ranksize - the number of clients nodes being used in this DR session.

ensight.UNIQUE_ID	
	This query returns a unique integer number every time it is called. This can be useful with things like obtaining a unique, private ‘filter’ flag for the ENS_EVENT_PYTHON event interface.

ensight.SPLINE_COUNT

	This query returns the number of splines currently defined. The value is returned as with the command driver queries: [errorcode, [number_of_splines]].

ensight.SPLINE_DATA	
	This query returns the acutal raw splince coordinates for a given spline number. For example: a = ensight.query(ensight.SPLINE_DATA, 1) returns the control points for spline number 1. The spline data itself has the following form: [errorcode,[num_points],[x0,y0,z0,x1,y1,z1...],[‘spline name’]]

ensight.TOOL_PARAMS *		
	Allows the user to query the current setting for the various EnSight data tools. The param1 value should be one of the following:	
	ensight.TOOL_CURSOR	Value is three floats: [x,y,z] the point
	ensight.TOOL_LINE	Value is six floats: [x0,y0,z0,x1,y1,z1] two points
	ensight.TOOL_PLANE	Value is twelve floats: [x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3] four points (must be rectangular and co-planar)
	ensight.TOOL_BOX	Value is fifteen floats: [ox,oy,oz,xx,xy,xz,yx,yy,yz,zx,zy,zz,sx,sy,sz] an origin point, three normal vectors (must be orthogonal) for the axis and three length values
	ensight.TOOL_CYLINDER	Value is seven floats: [x0,y0,z0,x1,y1,z1,rad] two points at the ends of the cylinder and the radius
	ensight.TOOL_SPHERE	Value is six floats: [x0,y0,z0,x1,y1,z1] two points that define the diameter (and major axis) of the sphere.
	ensight.TOOL_CONE	Value is seven floats: [x0,y0,z0,x1,y1,z1,cone_ang] two points at the ends of the cone and the angle at the apex.
	ensight.TOOL_REVOLUTION	Value is six + 2*N floats: [x0,y0,z0,x1.y1.z1,d0,r0,d1,r1,...] two point, one at the end and the other that defines the axis followed by a variable number of distance, radius pairs that define the profile

err = ensight.modify(param,(value))		
	This method is used to set various EnSight global parameters. The valid values for "param" are:	
	ensight.TEXTURE_IMAGE	
		In this case, value should be a tuple of the form (texure_index, image). Texture_index is an integer from 0 to 7 and image is an EnVe module image object.
	ensight.TEXTURE_BORDER_COLOR	
		In this case, value should be a tuple of the form (texure_index, (r,g,b,a)). Texture_index is an integer from 0 to 7, while (r,g,b,a) is a four valued float tuple that specifies a color+alpha value (all values are in the range [0,1]).
	ensight.TOOL_PARAMS	
		In this case, value should be a tuple of the form (tool_name, (tool_setting)). Tool_name is one of the options documented for TOOL_PARAMS for ensight.query() above (e.g. ensight.TOOL_CURSOR). Tool_setting is the value to change to change the tool location/settings to (see the TOOL_PARAMS section in the ensight.query() documentation for details on the individual tool formats.

ensight.refresh()	
	This method causes EnSight to redraw its graphics displays.

image = ensight.render([x=dx] [,y=dy] [,dpi=dpi] [,stereo=s] [,plots_only=po] [,invert_background=ib][,num_samples=ns])	
	This method causes EnSight to render the current image to one or more EnVe image objects. The return value is either a single EnVe image object by default. If the stereo keyword is set to 1, the return value will be a list of two image objects. The first is the left eye view and the second the right eye view. By default, the output image(s) will be the size of the current EnSight rendering window. The ‘x’ and ‘y’ keywords can be set to any desired output image size. The dpi keyword selects the dots per inch resolution of the output. In the current release, this keyword is ignored. When set to 1, the ‘plots_only’ keyword will cause only the EnSight plots to be drawn instead of both the plots and the geometry. When set to 1, the ‘invert_background’ keyword will cause the current background to toggle between black and white. This can be useful if the ultimate target of the image is a printer or perhaps an embedded document. The ‘num_samples’ keyword sets the number of passes of full scene anti-aliasing that should be used to render the output image. The default number of samples is 1 and the maximum number is 16.

value = ensight.query_points(pointlist [,parts=partlist] [,variables=varlist] [,time=t])	
	This function is a basic variable and mesh query mechanism. It takes a list of points as input. The list is a Python list of 3 value python lists. An example would be: [[1,2,0], [2,4,2]] for two points. The function will perform a query at each point in the list.

The “parts” keyword variable is optional. If not present, the currently selected parts will be used. The “parts” keyword can be set to the string “all” (e.g. parts=”all”) to select all parts in the current case, otherwise, “partlist” should be a Python list variable may include any mixture of string names, part numbers or part objects for all the parts that should be included in the query. For example: parts=[“mesh”,”field”,8] will include the parts named “mesh”, “field”, and part number 8 in the query.

The “variables” keyword variable is optional. If present, “varlist” should be a Python list variable that may include any mixture of string names, variable numbers or variable objects for all the variables that should be included in the query. For example: variables=[“velocity”,”pressure”,2] will include output for the variables named “velocity”, “pressure” and variable number 2 in the query.

The “time” keyword variable is optional. If not present, the query will be for the current time value in EnSight. The caller can also specify a specific time value using this keyword.

The return value is a list of output lists. One list for each input point. If no variables are specified, the query will be about the mesh structure and the list for each point will appear as:

[part_number, structuredpart_flag, closest_node, element_id]

The values are the part number the point was found in, a flag that is non-zero if the part is structured, the node id of the node closest to the point and the element id of the element that contains the point.

If a list of variables is specified, the returned value will be a list of lists, one for each variable. If the input variables were “pressure” and “velocity” (a scalar and a vector), the output (per point) would be:

[[pressure_value], [velocity_x, velocity_y, velocity_z]]

```
value = ensight.query_nodes(nodelist [,parts=partlist][,variables=varlist][,time=t])
```

This function is a basic variable and mesh query mechanism. It takes a list of node ids as input. An example would be: [500, 323] for two nodes. The function will perform a query at each node in the list.

The “parts” keyword variable is optional. If not present, the currently selected parts will be used. The “parts” keyword can be set to the string “all” (e.g. parts=”all”) to select all parts in the current case, otherwise, “partlist” should be a Python list variable may include any mixture of string names, part numbers or part objects for all the parts that should be included in the query. For example: parts=[“mesh”,”field”,8] will include the parts named “mesh”, “field”, and part number 8 in the query.

The “variables” keyword variable is optional. If present, “varlist” should be a Python list variable that may include any mixture of string names, variable numbers or variable objects for all the variables that should be included in the query. For example: variables=[“velocity”,”pressure”,2] will include output for the variables named “velocity”, “pressure” and variable number 2 in the query.

The “time” keyword variable is optional. If not present, the query will be for the current time value in EnSight. The caller can also specify a specific time value using this keyword.

The return value is a list of output lists. One list for each input node. If no variables are specified, the query will be about the mesh structure and the list for each node will appear as:

[part_number, structuredpart_flag, [node_x, node_y, node_z], [element_list]

	The values are the part number the node was found in, a flag that is non-zero if the part is structured, the coordinates of the node and a list of elements that reference that node. If a list of variables is specified, the returned value will be a list of lists, one for each variable. If the input variables were “pressure” and “velocity” (a scalar and a vector), the output (per node) would be: [[pressure_value], [velocity_x, velocity_y, velocity_z]]

`value = ensight.query_elems(elemlist [,parts=partlist][,variables=varlist][,time=t])`	
	This function is a basic variable and mesh query mechanism. It takes a list of element ids as input. An example would be: [232, 4733] for two elements. The function will perform a query at each element in the list. The “parts” keyword variable is optional. If not present, the currently selected parts will be used. The “parts” keyword can be set to the string “all” (e.g. parts=”all”) to select all parts in the current case, otherwise, “partlist” should be a Python list variable may include any mixture of string names, part numbers or part objects for all the parts that should be included in the query. For example: parts=[“mesh”,”field”,8] will include the parts named “mesh”, “field”, and part number 8 in the query. The “variables” keyword variable is optional. If present, “varlist” should be a Python list variable that may include any mixture of string names, variable numbers or variable objects for all the variables that should be included in the query. For example: variables=[“velocity”,”pressure”,2] will include output for the variables named “velocity”, “pressure” and variable number 2 in the query. The “time” keyword variable is optional. If not present, the query will be for the current time value in EnSight. The caller can also specify a specific time value using this keyword. The return value is a list of output lists. One list for each input element. If no variables are specified, the query will be about the mesh structure and the list for each element will appear as: [part_number, structuredpart_flag, element_type, failed_flag, [element_ijk_list], [node_list], [neighbor_element_list]] The values are the part number the element was found in, a flag that is non-zero if the part is structured, the type of the element, a flag that is non-zero if the element is currently “failed”, a list of ijk bounds for the element (if the part is structured), a list of the nodes this element references and a list of the elements that neighbor this element. If a list of variables is specified, the returned value will be a list of lists, one for each variable. If the input variables were “pressure” and “velocity” (a scalar and a vector), the output (per element) would be: [[pressure_value], [velocity_x, velocity_y, velocity_z]]

value = ensight.query_trace_paths([parts=partlist][,variables=varlist])	
	This function allows the user to directly query the points that make up particle traces. The output is similar to the output generated by the “test: trace_dump” command, except that it is returned as a Python variable insted of a text file on disk. The function takes a list of particle trace objects as input, along with an optional list of variable names.

	The “parts” keyword variable is optional. If not present, the currently selected parts will be used. The “parts” keyword can be set to the string “all” (e.g. parts=”all”) to select all parts in the current case, otherwise, “partlist” should be a Python list variable may include any mixture of string names, part numbers or part objects for all the parts that should be included in the query. For example: parts=[“mesh”,”field”,8] will include the parts named “mesh”, “field”, and part number 8 in the query. The “variables” keyword variable is optional. If present, “varlist” should be a Python list variable that may include any mixture of string names, variable numbers or variable objects for all the variables that should be included in the query. For example: variables=[“velocity”, ”pressure”, 2] will include output for the variables named “velocity”, “pressure” and variable number 2 in the query. *Note: in order to get a valid return for a given variable, it must be active, and have been activated prior to the creation of the trace.* The return value is a list of output lists. One list for each input part. The list for each part will contain one list for every trace in the part. If no variables are specified, the list for each trace will contain information about the temporal and spatial location of each point on the trace in the form: [x,y,z,t] The time value t is the resident time for the trace. If a list of variables is specified, the returned value will be a list per part of of lists per trace of lists, one for each variable. If the input variables were “pressure” and “velocity” (a scalar and a vector), the output (per point on a trace) would be: [[pressure_value], [velocity_x, velocity_y, velocity_z]] For example, in the case of two parts, each with 3 traces of 4 points each and the the varible keyword specifiying a vector and a scalar variable, the full output would look like: [#part 1 [#trace 1 [[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]]], #trace 2 [[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]]], #trace 3 [[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]],[[vx,vy,vz],[s]]]], #part 2 [...]]

ensight.query_parts([parts=partlist][,client=client][,variables=varlist])	

	This method returns basic information about parts as a Python list. The “parts” keyword variable is optional. If not present, the currently selected parts will be used. The “parts” keyword can be set to the string “all” (e.g. parts=”all”) to select all parts in the current case, otherwise, “partlist” should be a Python list variable may include any mixture of string names, part numbers or part objects for all the parts that should be included in the query. For example: parts=[“mesh”,”field”,8] will include the parts named “mesh”, “field”, and part number 8 in the query.
	
	The “variables” keyword variable is optional. If present, “varlist” should be a Python list variable that may include any mixture of string names, variable numbers or variable objects for all the variables that should be included in the query. For example: variables=[“velocity”, ”pressure”, 2] will include output for the variables named “velocity”, “pressure” and variable number 2 in the query.
	
	The return value is a list of output lists, one list for each selected part. The list for each part can have one of three structures, depending on the setting of the ‘client’ keyword and the presence of the variables keyword. If the variables keyword is present, the return value is a list of lists, one per parts. Each list has the following structure:
	`[partID, [vlist]]`
	The vlist list has one value for each variable being queried. The value is 1 if the variable exists, is active and has some defined values for the given part. The value is -1 if the variable not active or not defined for the given part. -2 is returned if the part does not contain the variable or the variable is not defined. Other values are returned for invalid input parameters.
	
	By default, the client keyword is 0 and results in a server query, much like the Query->Show Information->Part menu selection.Client side parts (e.g. groups, pathlines, vector arrows) will not be included in this output.If the client keyword is set to the value ‘1’, the query will be performed on the client. This is a much faster query and it includes all parts, but it returns less detailed information than the server query.
	
	In the server query case (client=0), the list for each part has the following structure:
	
	`[partID, part_type, [x0,y0,z0], [x1,y1,z1], num_nodes, elemlist, is_selected, “partname”, is_structured, ijkbounds, casenum, parentpartlist]`
	
	In the client query case (client=1), the list for each part has the following structure:
	
	`[partID, part_type, is_selected, “partname”, is_structured, casenum, parentpartlist]`

	In both cases, the named fields are defined as: partID = the part ID part_type = the EnSight part type (e.g. model, clip, etc) (int) part_type will be one of the following values: ensight.PART_MODEL, ensight.PART_CLIP_PLANE, ensight.PART_CONTOUR, ensight.PART_DISCRETE_PARTICLE, ensight.PART_FRAME, ensight.PART_ISO_SURFACE, ensight.PART_PARTICLE_TRACE, ensight.PART_PROFILE, ensight.PART_VECTOR_ARROW, ensight.PART_ELEVATED_SURFACE, ensight.PART_DEVELOPED_SURFACE, ensight.PART_MODEL_EXTRACT, ensight.PART_MODEL_CUT, ensight.PART_MODEL_BOUNDARY, ensight.PART_ISO_VOLUME, ensight.PART_BUILT_UP, ensight.PART_TENSOR_GLYPH, ensight.PART_FX_VORTEX_CORE, ensight.PART_FX_SHOCK, ensight.PART_FX_SEP_ATT, ensight.PART_MATERIAL_INTERFACE, ensight.PART_POINT, ensight.PART_AXI_SYMMETRIC, ensight.PART_MODEL_MERGE, ensight.PART_MULT x0,y0,z0 = the part minimum x,y,z coords x1,y1,z1 = the part maximum x,y,z coords num_nodes = the number of nodes in the part elemlist = a list of element types and the number of elements each type. For each element type in the part, there is a two element list in elemlist of the form: ['elemtype', num], that defines the number of elements of the specified type. elemtypes ar text strings like 'Tria3' or 'Hexa8'. is_selected = an integer that is non-zero if the part is currently selected partname = the name of the part as a string is_structured= an integer that is non-zero if the part is structured. If non-zero, the ijkbounds list will be present. ijkbounds = this list is only present if the part is structured. It consists of three, three element lists and has the form: [[i_min, i_max, i_step, i_limit], [j_min, j_max, j_step, j_limit], [k_min, k_max, k_step, k_limit]]. These are the current min, max and step size along the i, j and k axis for the structured part as well as the maximum (limit) of the individual axis bounds. casenum = the case number to which this part belongs parentpartlist= a list of part numbers that are the parents to this part. This list may be empty. It will be non-empty for derived parts (e.g. clips, contours, etc).

`ensight.query_cases()`
This method returns basic information about the current EnSight cases as a Python list. The returned list has the following format: [current_case_number,[case1_num,”case1_name”],[case2_num,”case2_name”]...] The first item is the current case number (zero based). There will be one subsequent item for each current case. Each of those items is a list containing the number for that case and its name.

`err=ensight.query_xy_create(title,xtitle,ytitle,data[,segments=seg])`

	This function creates a new xy query in EnSight. It is similar to reading a query from an external file, except the data is passed from Python. "title", "xtitle" and "ytitle" are strings that specify the textual labels for the curve, x and y axis. "data" is a list of lists that takes the form: [[x0,y0],[x1,y1],[x2,y2],...] That is a list of two element lists which contain two float values each that are interpreted as an X,Y point in space. By default, the data is plotted as a connected line. The optional "segments=seg" keyword allows the caller to specify a "seg" variable which is a list of integers. Each integer is the length of a connected line of points. For example: [10,24,8] specifies that the first 10 points should be connected as a polyline, the next 24 as separate polyline and the remaining 8 as a separate polyline. The sum of the "seg" list integers must be the same as the number of points in the "data" list. A return value of 0 means the creation was successful, -1 means there were issues in the format of the input arguments, -2 suggests an out of memory condition and -3 means that the query was created, but that some values in the "data" list were missing (and set to 0.0).

`ensight.qtparent()`	
	This method returns 'None' on X11 platform and a QWidget object under Windows. It is used in conjunction with ensight.reparent() to ensure that Python Qt windows can interact properly with the existing EnSight windows.

`ensight.version()`	
	This method returns a list of the form: ['vers', 'mode', 'arch', 'cei_home', 'server_vers', 'sos_vers', licenseinfo] vers = the version of EnSight currently running mode = 'gold' or 'standard' or 'lite' or 'demo' arch = the system architecture (output of cei_arch) server_ver = the version of the currently connected server (if any) sos_ver = the version of the currently connected sos (if any) licenseinfo = a dictionary of the current ensight license information

`ensight.checkabort([throw_exception])`	
	In the EnSight Python editor, there is an option to set an “abort” flag. When this method is called, two things happen. First, all the pending GUI events are processed. Second, if the user has selected the abort menu option, this method resets the abort state and returns a non-zero integer. If the abort menu has not been selected, this method returns the value 0. If throw exception is passed as an argument and is non-zero, if the function would return a non-zero value, it will instaed throw a Python exception.

`ensight.batch()`	
	This method returns a non-zero integer if the EnSight client is running in batch mode.

ensight.exit()	
	This method causes EnSight to shut down the current EnSight session.

The Python native command language interface

As described in the previous section, the ensight.sendmesg() function can be used to execute command language from Python. The command language takes the form of a Python string or list of strings. There exists a collection of modules that implement a more natural Python (“native”) interface to the command language. These modules follow the form of the EnSight command language syntax. All command language commands have the fundamental form:

```
category: command <args>
```

For example:

```
view_transf: rotate 9.657229e+000 2.282249e+001 0.000000e+000
```

has a category of ‘view_transf’, a command of ‘rotate’ and arguments of ‘9.657229e+000 2.282249e+001 0.000000e+000’. The native command language interface maps the category and command names to modules under the ‘ensight’ module. For example, the above command is equivalent to the following line of Python code:

```
ensight.view_transf.rotate('9.657229e+000 2.282249e+001 0.000000e+000')
```

Additionally, the native interface will try to map data types to <args> strings. So the following lines of Python code are all equivalent:

```
ensight.view_transf.rotate(9.657229e+000,2.282249e+001,0.000000e+000)
```

```
ensight.view_transf.rotate([9.657229,2.282249e+01,0])
```

```
a=[9.657229,2.282249e+01,0]
```

```
ensight.view_transf.rotate(a)
```

The use of Python native variables makes this interface much more flexible for complex scripting.

The native interface handles the “select_begin”/”select_end” commands very differently. The command language includes structures like:

```
part: select_begin
 1 2 3 4 5 6 7 8
part: select_end
part: select_byname_begin
"(CASE:Case 1)Block ID 7 - HEX" "(CASE:Case 1)Block ID 8 - HEX"
"(CASE:Case 1)Block ID 9 - HEX" "(CASE:Case 1)Block ID 10 - HEX"
"(CASE:Case 1)Block ID 11 - HEX" "(CASE:Case 1)Block ID 1 - HEX"
part: select_byname_end
```

The native Python interface is aware of the list nature of the “select_begin”/”select_end” commands and maps them to a single command (the “select_begin”). The above lines translate to two lines of Python:

```
ensight.part.select_begin(1,2,3,4,5,6,7,8)
ensight.part.select_byname_begin("(CASE:Case 1)Block ID 7 -
HEX","(CASE:Case 1)Block ID 8 - HEX","(CASE:Case 1)Block ID 9 -
HEX","(CASE:Case 1)Block ID 10 - HEX","(CASE:Case 1)Block ID 11 -
HEX","(CASE:Case 1)Block ID 1 - HEX")
```

The “*_end” functions are not necessary. Additionally, a list-based syntax is also supportted, so the following three lines are also equivalent:

```
ensight.part.select_begin([1,2,3,4,5,6,7,8])
list = ["(CASE:Case 1)Block ID 7 - HEX",
        "(CASE:Case 1)Block ID 8 - HEX",
        "(CASE:Case 1)Block ID 9 - HEX",
        "(CASE:Case 1)Block ID 10 - HEX",
        "(CASE:Case 1)Block ID 11 - HEX",
        "(CASE:Case 1)Block ID 1 - HEX"]
ensight.part.select_byname_begin(list)
```

In all cases where the command language uses a “*_begin”/”*_end” syntax with a collection of items (numerical or strings), a Python command strucure similar to the above may be used.

Conversion of command language scripts to the native interface

In previous versions of EnSight, functions were provided to convert selections of command language to ensight.sendmesg() format. With the native interface, these have been replaced with two menu options in the EnSight Python script editor. Under the ‘Edit’ menu, there is an option ‘Convert selection to sendmesg()’ which converts the currently selected text in the editor into the ‘ensight.sendmesg()’ form. More importantly, there is a ‘Convert selection to native Python’ option. This function will convert command language to the native interface described here. One can convert a .enc file by loading it into the script editor window, selecting all of the text and converting it to native Python. The result can be saved as a ‘.py’ file. An example of the conversion results would be:

```
VERSION 8.26
part: select_default
part: modify_begin
part: elt_representation 3D_feature_2D_full
part: modify_end
data: binary_files_are big_endian
data: format case
data: shift_time 1.000000 0.000000 0.000000
data: replace c:/Program Files/CEI/ensight100/data/cube/cube.case
view_transf: rotate -6.277198e+001 6.945956e-001 0.000000e+000
part: select_all
clip: begin
clip: value MID-RANGE
clip: domain intersect
clip: tool xyz
clip: end
clip: create
part: select_begin
 2
part: select_end
view_transf: rotate 1.904619e+001 3.095920e+001 0.000000e+000
part: modify_begin
clip: value -0.425000
clip: tool xyz
part: modify_end
view: hidden_surface ON
variables: activate temperature
part: select_all
part: modify_begin
part: colorby_palette temperature
part: modify_end
```

Translates into the following native Python code:

```
ensight.sendmesgoptions(version=8.26)
ensight.part.select_default()
ensight.part.modify_begin()
ensight.part.elt_representation("3D_feature_2D_full")
ensight.part.modify_end()
ensight.data.binary_files_are("big_endian")
ensight.data.format("case")
ensight.data.shift_time(1.000000,0.000000,0.000000)
ensight.data.replace("c:/Program Files/CEI/ensight100/data/cube/cube.case")
ensight.view_transf.rotate(-6.277198e+001,6.945956e-001,0.000000e+000)
ensight.part.select_all()
ensight.clip.begin()
ensight.clip.value("MID-RANGE")
ensight.clip.domain("intersect")
ensight.clip.tool("xyz")
ensight.clip.end()
ensight.clip.create()
ensight.part.select_begin(2)
ensight.view_transf.rotate(1.904619e+001,3.095920e+001,0.000000e+000)
ensight.part.modify_begin()
ensight.clip.value(-0.425000)
ensight.clip.tool("xyz")
ensight.part.modify_end()
ensight.view.hidden_surface("ON")
ensight.variables.activate("temperature")
ensight.part.select_all()
ensight.part.modify_begin()
ensight.part.colorby_palette("temperature")
ensight.part.modify_end()
ensight.sendmesgoptions(version=0)
```

This language conversion is actually performed by the class ‘Cmd2Py’ in the Python module: cei.cmd2py. A quick study of the cmd2py.py file will give an overview of the conversion process and what is and what is not handled by the conversion. Some command language cannot be converted automatically. Files that include things like command language loops cannot be converted automatically and the conversion system will insert comments detailing the failures. Examples that use other command language features that cannot be directly translated (e.g. ‘$’ variables) will be converted to ensight.sendmesg() formatted lines as a fallback. In most cases, users are better off converting such sections of code into native Python in any case. A command line program: ‘cmd2py’ is also provided to aid in the conversion of files outside of EnSight.

Usage notes on the native Python interface

The native interface has the same limitations listed in the “Limitations of the EnSight Python Interface”. In general, all of the command language is supported with a few exceptions.

One exception is the ‘test:’ commands. These commands are not supportted by the native interface (i.e. there is no ‘ensight.test’ module). They can of course be issued using the ensight.sendmesg(“test:...”) interface.

There are a number of commands in the EnSight command language that are not valid Python names. A few examples include:

```
function: #_of_levels 5
annotation: 3d_label_size 10.0
command: print “hello”
viewport: raise
```

The reasons a name might be invalid include:

1) name includes an invalid character (e.g. “#”)
2) name begins with a digit (e.g. “1”)
3) name is a Python reserved word (e.g. “raise”)

These are transformed using the following rules:

1) “#” characters are replaced with the text “number”
2) names that start with a digit are prefixed with an “_”
3) names that are the same as a Python reserved word are prefixed with an “_”

The previous examples become:

```
ensight.function.number_of_levels(5)
ensight.annotation._3d_label_size(10.0)
ensight.command.print("hello")
ensight.viewport._raise()
```

There are a few obsolete EnSight command language functions that are not supportted. One example would be:

```
part: symmetry_1_1_-1
```

Such functions can still be executed using ensight.sendmesg().

The native interface methods do not include the ‘exception’, ‘record’, ‘display’ or ‘version’ keywords like ensight.sendmesg(). However, they do honor the values set by ensight.sendmesgoptions(). That function can be used to cause the code that uses the native interface do be executed under the particular rules of a specific command language version or to throw exceptions on errors for instance.

EnSight Python events

EnSight has a mechanism to execute Python code when various events occur in the main program. The module provides methods to add a callback and remove one. Each callback consists of a Python object, the name of a method on the object and a reason to be called back. An example Python class that registers itself for one rendering callback and then removes itself would be:

```
class cb_class:
  def register(self):
    self.ID = ensight.addcallback(self,"callback",ensight.ENS_EVENT_PRERENDER)
  def callback(self,value):
    print "Called back from EnSight", value
    ensight.removecallback(self.ID)
example = cb_class()
example.register()
```

Note that "callback" in the addcallback() method specifies the name of the function in the object to call. The value in the callback "value" argument vary depending on the specific event it is tied to, but it will always be a tuple that starts with the original event name passed in addcallback(). Another example, using the ENS_EVENT_PYTHON interface might be:

```
class py_cb_class:
  def __init__(self):
    self.filter = ensight.query(ensight.UNIQUE_ID)
    self.ID1 = ensight.addcallback(self,"fil_cb",ensight.ENS_EVENT_PYTHON,self.filter)
    self.ID2 = ensight.addcallback(self,"def_cb",ensight.ENS_EVENT_PYTHON)
  def fil_cb(self,value):
    print "Filtered callback ", self.filter, value
    return 0
  def def_cb(self,value):
    print "Default callback ", value
    return 0
  def test(self):
    print “immediate,filtered:”, ensight.sendevent([“hello”],filter=self.filter)
    print “deferred:”, ensight.sendevent([“goodbye”],deferred=1)
example = py_cb_class()
example.test()
```

In this example, take careful note of the position and order of the generated output to see the effects of deferred callbacks.

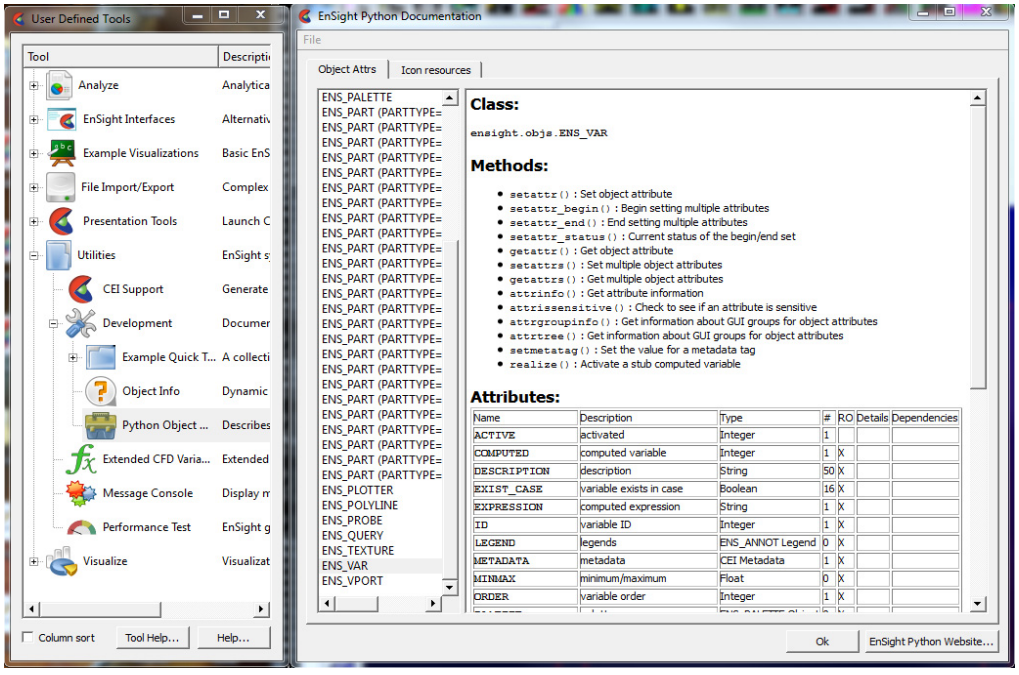
EnSight Python Code Methods

`ID = ensight.addcallback(object,"methodname",event_type[,timeout|filter])`		
	This method registers the specified method name will be called on the passed object when the specified event_type occurs in EnSight. The function returns an ID number that can be passed to removecallback() to unregister the callback. Currently defined values for event_type:	
	`ensight.ENS_EVENT_ALL`	The method will be called for all EnSight events. The callback function value will reflect the actual event type for which the callback was invoked.
	`ensight.ENS_EVENT_QUIT`	EnSight is about to exit.
	`ensight.ENS_EVENT_SOLUTION_TIME`	The current solution time has changed. The new time is passed as the value.
	`ensight.ENS_EVENT_PRERENDER`	EnSight is about to redraw its current displays.
	`ensight.ENS_EVENT_POSTRENDER`	EnSight has just redrawn its current displays.
	`ensight.ENS_EVENT_PRECOMMAND`	EnSight is about to execute the command language string passed as the value.
	`ensight.ENS_EVENT_POSTCOMMAND`	EnSight has just executed the command language string passed as the value.
	`ensight.ENS_EVENT_TIMEOUT`	A periodic time has occurred (see below).
	`ensight.ENS_EVENT_PARTSELECT`	The current part selection has changed to the tuple of part numbers passed as value.
	`ensight.ENS_EVENT_PART`	A part has either been created or destroyed. A value of 0 is passed on construction and 1 is passed on destruction. A value of 3 is passed when a part is renamed.
	`ensight.ENS_EVENT_CASE`	A case has either been created, renamed, made current or destroyed. A value of 0 is passed on construction, 1 is passed on destruction, 2 is passed if the current case is made current and 3 is passed if a case is renamed.
	`ensight.ENS_EVENT_QUERY`	A query in ensight has been created, destroyed or updated. Two values are passed. The first selects the type of the query. If the first value is: ensight.ENS_EVENT_QUERY_PROBE, the second value will be 0 and the current probe information has been updated. If the first value is: ensight.ENS_EVENT_QUERY_DATA the value can be 0 if a query has been created or 1 if it has been destroyed.
	`ensight.ENS_EVENT_VARIABLE`	A variable has either been created/activated or destroyed/deactivated. A value of 0 is passed on construction and 1 is passed on destruction.

Special event Types:	
These events come from different sources and the callback functions may have slightly different behavior from the basic event types.	
Python events:	
`ensight.ENS_EVENT_PYTHON`	This event allows multiple Python objects running inside of the EnSight Python interpreter to pass information to each other. A callback function that is associated with this event type will be passed a Python list as its value (see: the method ensight.sendevent()). The return value of this callback function is important. If it has the integer value 1, no additional `ENS_EVENT_PYTHON` callbacks will be called with the value. If the callback returns the value 0, any remaining `ENS_EVENT_PYTHON` callbacks will also be called with the same value. When this event is specified, the optional 'filter' parameter to the ensight.addcallback() is used. The parameter registers a "tag" with the callback that specifies that only the callbacks generated by a sendevent() call with that same tag be called. By default, the filter values are set to zero in both functions which results in a broadcast of the python list to all callbacks. The application can specify a unique 'tag' number to send private messages. See ensight.sendevent() for more details.
Low Level CVF (device) events:	
EnSight's rendering windows are based on a framework called CVF. This framework provides an abstraction for all user input events. These event types give a Python callback the opportunity to see these low-level device events. It also gives the callback the opportunity to suppress the handling of these events by the normal EnSight handlers. This can be useful for applications that decide they want specify control over user input in the graphics windows or want to override the default EnSight behavior for types of interaction. If the callback for one of these events returns the integer value 1, the event will not be passed on to EnSight. A return value of 0 will allow normal EnSight event processing to occur.	
`ensight.ENS_EVENT_MOUSE_BUTTON_DOWN` The passed value will be the list: [ENS_EVENT_MOUSE_BUTTON_DOWN, button, buttonstate, modifiers, x, y]	
`ensight.ENS_EVENT_MOUSE_BUTTON_UP` The passed value will be the list: [ENS_EVENT_MOUSE_BUTTON_UP, button, buttonstate, modifiers, x, y]	
`ensight.ENS_EVENT_MOUSE_DOUBLE_CLICK` The passed value will be the list: [ENS_EVENT_MOUSE_DOUBLE_CLICK, button, buttonstate, modifiers, x, y]	
`ensight.ENS_EVENT_MOUSE_MOTION` The passed value will be the list: [ENS_EVENT_MOUSE_MOTION, buttonstate, modifiers, x, y]	
`ensight.ENS_EVENT_WHEEL_MOTION` The passed value will be the list: [ENS_EVENT_WHEEL_MOTION, val, dir, modifiers, x, y]	

	ensight.ENS_EVENT_6D_BUTTON_DOWN The passed value will be the list: [ENS_EVENT_6D_BUTTON_DOWN, tracker]
	ensight.ENS_EVENT_6D_BUTTON_UP The passed value will be the list: [ENS_EVENT_6D_BUTTON_UP, tracker]
	ensight.ENS_EVENT_6D_VALUATOR The passed value will be the list: [ENS_EVENT_6D_VALUATOR, valuator, value]
	ensight.ENS_EVENT_DRAWABLE_RESIZE The passed value will be the list: [ENS_EVENT_DRAWABLE_RESIZE]
	ensight.ENS_EVENT_DRAWABLE_EXPOSE The passed value will be the list: [ENS_EVENT_DRAWABLE_EXPOSE
	ensight.ENS_EVENT_DRAWABLE_SHOW The passed value will be the list: [ENS_EVENT_DRAWABLE_SHOW]
	ensight.ENS_EVENT_DRAWABLE_HIDE The passed value will be the list: [ENS_EVENT_DRAWABLE_HIDE]
	ensight.ENS_EVENT_DRAWABLE_FOCUSIN The passed value will be the list: [ENS_EVENT_DRAWABLE_FOCUSIN]
	ensight.ENS_EVENT_DRAWABLE_FOCUSOUT The passed value will be the list: [ENS_EVENT_DRAWABLE_FOCUSOUT]
	ensight.ENS_EVENT_DRAWABLE_ENTER The passed value will be the list: [ENS_EVENT_DRAWABLE_ENTER]
	ensight.ENS_EVENT_DRAWABLE_LEAVE The passed value will be the list: [ENS_EVENT_DRAWABLE_LEAVE]
	ensight.ENS_EVENT_DRAWABLE_SAVE The passed value will be the list: [ENS_EVENT_DRAWABLE_SAVE]
	ensight.ENS_EVENT_DRAWABLE_RESTORE The passed value will be the list: [ENS_EVENT_DRAWABLE_RESTORE]
	The x and y values are the location of the cursor at the time of the event. Generally, the button value will be one of (ensight.ENS_BUTTON_LEFT, ensight.ENS_BUTTON_RIGHT, ensight.ENS_BUTTON_MIDDLE, ensight.ENS_BUTTON_WHEEL), while buttonstate is the arithmetic or of these values together for the current state of all the buttons. The modifiers values will be the arithmetic or of none or some of the following values: ensight.MODIFIER_CTRL, ensight.MODIFIER_SHIFT, ensight.MODIFIER_ALT.
	Timer events
	ensight.ENS_EVENT_TIMEOUT This is a special case in that it requires an extra "timeout" argument to addcallback(). Registering for this event type schedules a periodic callback to the Python code from EnSight every "timeout" seconds (timeout is a float). ENS_EVENT_ALL callbacks are not called for ENS_EVENT_TIMEOUT events.

err = ensight.removecallback(ID)	
	This method removes the previously registered callback function.

suppressed = ensight.sendevent(python_list [,filter=f][,deferred=d])	
	This method works in conjunction with ensight.ENS_EVENT_PYTHON. When this function is called from the EnSight Python interpreter, all of the callback functions that were associated with ENS_EVENT_PYTHON are called with the Python list object passed to this function as a parameter. If a callback returns the value of 1, then the list is not passed to any additional callback functions of this type and sendevent() returns 1. This mechanism is used to allow multiple objects running within the EnSight Python interpreter to communicate with each other and pass data, messages, etc back and forth within EnSight. By default, this is a synchronous call, so callback functions that themselves call sendevent() must take care to ensure that their callback function is reentrant. The user may also specify the 'deferred=' keyword to make the callback asynchronous. If "deferred" is set to 1, this function will return immediately, but will queue up the callback for later execution. A the next point where EnSight becomes "idle" the callbacks will be executed. This can be very useful in breaking circular callback loops. By default, the python list object is sent to all methods registered for the ENS_EVENT_PYTHON callback. The 'filter=' keyword can be used to select a sepecific subset of registered methods be called instead of all of them. The filter keyword works in conjunction with the filter parmeter to ensight.addcallback() to create private communications channels. The ensight.query(ensight.UNIQUE_ID) call can be used to generate guaranteed unique numbers that can be used for filter values at runtime to avoid colliding with other applets which may also use this mechanism.
	

6.2 EnSight object API

Starting in EnSight 9.0, CEI has been developing a new Python API based on the intrinsic objects inside of the EnSight core. This API tragets applications developers that wish to build more complex applications on top of the EnSight Qt and Python interfaces. This API is utilized by EnSight itself internally and by applications such as EnSight CFD. The API is much larger than the previous API and the documentation would not fit easily into these documents. Furthermore, the API is not yet complete and CEI reserves the right to change anything in the API without warning at this point in time. Many folks have asked for access to this API, even though it is not yet complete. CEI does not offer formal support for any of the Object API interfaces at this point in time. We will take bug reports, but due to the nature of the API, we are not providing normal support services.

CEI provides two locations for information on the API. First, in the user-defined tools, there is a tool located under Utilities/Development named “Python Object API”. This tool allows the user to browse the various classes available and their attributes/methods. The tool is “self-documenting” in that all the information it displays is generated from the core objects at runtime. There is also a website: https://sites.google.com/a/ensight.com/ensight-python-api/ that contains additional information about the various APIs. The website is dyanmically updated and interested parties can suscribe to changes via RSS feeds.

6.3 Python EnVe module interface

The EnSight Python interpreter includes the EnVe 3.0 module as well and can be imported as:

```
import enve
```

EnSight uses this module internally and in some cases it may have imported it already. The 'enve' module defines a pair of Python objects that encapsulate a movie and an image as well as a few helpful secondary functions for things like listing the system installed UDILs.

The EnVe Movie object

The movie object is created in READ or WRITE mode. In normal operation, the various movie attributes are set after creation and then the object is 'open()'ed to begin the I/O process. Once the file is open, some attribute values will change to match the actual values in the files and some attributes will become readonly until the file is closed. Movie objects are associated with a filename. They contain a number of images that can be read as well as attributes like dimensions, frames per second and stereo.

The movie object is used to open and read from or write to animation files. A simple example of reading a movie in one format and writing it to an EVO file (note: no error checking is done in the example) is shown here:

```
mov = enve.movie(enve.MOVIE_READ)
mov.filename = "filetoread.mpg"
mov.open()
evo = enve.movie(enve.MOVIE_WRITE)
evo.filename = "myoutputfile"
evo.format = "EVO"
evo.options = "Compression RLE"
evo.addcount(mov) # how many frames will be added
evo.open()
evo.append(mov)
evo.close()
mov.close()
```

EnVe Module Code Methods

`x = enve.movie(enve.MOVIE_READ|enve.MOVIE_WRITE)`	
	Creates an empty movie object in read or write mode. No actual I/O operations will be started until the “open()” method is called. The proper usage is to create the movie object, set attributes on it (e.g. filename, image size, number of frame in the case of writing) and then open() the file. At that point, the append() and getframe() methods will work. When done accessing image frames, the file should be close()ed and the object can be destroyed.

`err = x.addcount(N|image|movie)`
Write movies must know in advance how many frames will be added before the file is open()ed. This function is used to add frame counts to the movie. The input parameter can be an integer (number of frames) an image object (adds one frame) or another movie (open() and in READ mode, which adds the number of frames in the movie). In the latter two cases, the target movie may change its dimensions or frame rate to match the added objects. This will happen if the current dimensions or fps are <= 0. The return value is the number of frames, or -1 on error. x.nframes is updated to reflect this as well. If addcount() is only called with an integer number of frames, the caller must also set the 'dims' attribute to set the size of the output movie in pixels.

`err = x.resetcount()`
This function resets the current nframes count on a WRITE movie to 0. This can be called if the movie is not currently open.

`err = x.open()`
This method opens the physical movie file. In the case of a WRITE file, it will begin the encoding process.

`err = x.append(image|movie[,object=obj][,method=meth][,smooth=sm])`
If a WRITE movie is open(), this method is used to physically add an image or the frames in a movie to the current movie. The input frames will be scaled to match the dimensions of the target. If the caller specifies the OBJECT and METHOD keywords to an object reference and the name of a method, this function will periodically call 'object.method(n)' where n is the number of the frame currently being worked on. This method should return 1 if the operation is to be aborted or 0 if processing should continue. By default, this method uses nearest neighbor sampling to scale images. If a higher quality scaling is desired, set the SMOOTH keyword to the value enve.CVF_SCALE_SMOOTH to select box filtered scaling.

`image = x.getframe(framenum)`
This method extracts a frame from a movie and returns a list of image objects. There may be one or two images returned (two in the case of a hardware stereo source). If two images are returned, the first is the left eye image and the second is the right eye image. The framenum is [0,x.nframes-1]. The returned frame is intensity and repeat adjusted as specified by those attributes. This method can only be used on a READ movie.

`err = x.close()`
This method completes all I/O operations with a given movie. The physical file on disk will be valid after this method is called.

`str = x.errstr()`	
	If any of the methods return an error (-1), a string describing the error will be stored in the movie object. These error strings can be accessed via this method.

`print x`	
	Prints basic information about the movie x.

```
x.attr
x.attr = y
```

These get and set any of the various movie object attributes. **Movie attributes:**	
`__members__`	Returns a list of the attributes this object supports. Readonly list.
`__methods__`	Returns a list of the methods this object supports. Readonly list.
`filename`	The filename to use. Read/write string.
`stereo`	This attribute is non-zero if the movie supports HW stereo (on read) or if the movie has been set to output HW or anaglyph stereo (on write). Read/write int.
`fps`	The framerate for movie playback in frames per second. Read/write float.
`dims`	The dimensions of the movie frames in pixels. Read/ write tuple of two ints: (dx, dy)
`tiling`	If the output movie should be written as an MTM file, this attribute specifies the number of tiles in the X and Y axis. Read/write tuple of two ints: (dx, dy)
`format`	This attribute specifies the format for the movie. See: enve.codecs() for a list of formats. Read/write string.
`framerange`	For a read movie, this attribute allows one to select a subset of the input frames to read. For example, (10,20) will cause the movie to only output frames 10 through 20 inclusive (11 frames). Note that framerange numbers start at 0 and run through x.realframes-1. Read/write tuple of two ints: (start, end)
`nframes`	For a read movie, this is the effective number of frames in the file (and the valid range for getframe(X)). It is equal to: frameend-framestart+repeatstart+repeatend. Readonly int.
`realframes`	The number of actual physical images in a movie file. Readonly int.
`options`	Format specific options in the form: "op1 value op2 value ...". See enve.codecs() for a list of format options. Read/write string.
`intensity`	When reading from a file, the frames may have their intensity scaled linearly from the first to the last frame. This attribute allows this scaling to be set. The default is (1.0,1.0) or no intensity changes. Read/write tuple of two floats: (start, end)

	repeat	When reading from a file, the first and last frames may be repeated a number of times. This attribute allows the number of additional times those frames appear. For example, (5,3) will cause the first frame to be repeated 5 additional times (6 total) and the last frame 3 times (4 total). As a result, nframes will report 8 more frames in the movie. Note that the intensity interpolation includes these repeated frames. Read/write tuple of two ints: (start, end)
	anaglyph	If an output movie has stereo set, this attribute allows that stereo to be in anaglyph form. Valid values include: `enve.MOVIE_ANAGLYPH_NONE` - use HW stereo (shutter glasses) enve.MOVIE_ANAGLYPH_REDBLUE enve.MOVIE_ANAGLYPH_BLUERED enve.MOVIE_ANAGLYPH_REDCYAN `enve.MOVIE_ANAGLYPH_CYANRED` Read/write int.
	flip	When reading from a file, flip the frames over the X or Y axis. Valid values are formed by oring: `enve.CVF_FLIP_XAXIS` and/or `enve.CVF_FLIP_YAXIS` Read/write int.

The EnVe Image object

The image object is a simple 2D array of pixels. An image can be in a number of formats:

```
enve.CVF_IMG_FMT_A        - alpha only image
enve.CVF_IMG_FMT_L        - luminance only image (grayscale)
enve.CVF_IMG_FMT_LA       - luminance+alpha image
enve.CVF_IMG_FMT_RGB      - red,green,blue three channel image
enve.CVF_IMG_FMT_RGBA     - red,green,blue,alpha four channel image
```

EnVe image object processing operations

`x = enve.image()`	
	Create a new image object. The default image is 100 by 100 in size and in RGB mode.

`str = x.ppm()`	
	This method returns a string that is the representation of the image as an ASCII PPM file. This can be useful in interfacing to systems like Qt.

```
image = x.subrect((int,int),(int,int))
image = x.subrect()
```

This method extracts a rectangle of pixels from the image and returns a new image. The first tuple is the offset into the image and the last specifies the dx,dy in pixels. In the second form, the method simply clones the source image.

```
x.chromakey(incolor,tolerance,alpha)
```

This method scans the input image and compares the color of each pixel to the input pixel color. If every channel is within 'tolerance' of the target pixel, the alpha channel for that pixel is replaced by the input alpha value. incolor is a tuple of 3 integers (R,G,B). tolerance is a tuple of 3 integers that are the tolerance in R,G,B.

```
x.flip(int)
```

This method will flip an image over one or more axis. The int parameter formed by oring together the following: enve.CVF_FLIP_XAXIS and enve.CVF_FLIP_YAXIS, selects the operation.

```
x.swizzle(swiz,mask)
```

This method allows the value of any channel of the image to come from any other channel channel of the image. The array swiz selects the input channel for each output channel. For example, a swiz of (0,1,2,3) will result in no image change. A swiz of (2,1,0,0) will swap the red and blue channels and place the input red channel in the output alpha channel as well. The mask array allows individual output channels to be enabled or disabled for writing. The value in the mask array must be non-zero for the output channel to be writable. swiz and mask are both tuples of four integers. With some formats, not all values are used.

```
x.colormath(rw,gw,bw,aw)
```

This method performs simple linear algebra on an image. The math is performed at floating point resolution and the results clamped to the range [0,255] before being output. rw,gw,bw and aw are each tuples of 5 floats and each represent the coefficients of the linear transform for a given output component.

The math implemented is as follows:

```
R' = R*rw[0]+G*rw[1]+B*rw[2]+A*rw[3]+rw[4]
G' = R*gw[0]+G*gw[1]+B*gw[2]+A*gw[3]+gw[4]
B' = R*bw[0]+G*bw[1]+B*bw[2]+A*bw[3]+bw[4]
A' = R*aw[0]+G*aw[1]+B*aw[2]+A*aw[3]+aw[4]
```

```
x.setpixel(xo,yo,pixelvalue)
```

This method will set the color of the pixel at position xo,yo in the image to be the color specified by the list pixelvalue. Pixelvalue is a list of integer values between 0 and 255 (inclusive). There should be at least as many elements on the list as there are image components.

`pixelvalue = x.getpixel(xo,yo)`	
	This method will returns as a list, the current color at pixel xo,yo in the image. See setpixel() for a description of the pixelvalue.

`s = x.tostring()`	
	This method will return the pixel data in the image as a raw binary Python string. There is one character in the string for each component of each pixel. For example, if the image were 100x100 and in RGB format, the resulting string would be 30000 bytes in length. Note the the image dimensions and pixel format are not encoded in the string and that the string is actually raw binary data. This method can be used with packages such as matplotlib or numpy which provide mechanisms for interpreting such raw strings.

`x.fromstring(s, dx, dy, format)`	
	This method is the inverse of tostring(). It constructs the contents of the image x from the pixel data in the raw binary string s. It is assumed that the string source was something like that output by the enve.image.tostring() method. The dx, dy and format arguments specify the size and pixel format the image x should take. This method can be used with packages such as matplotlib or numpy which provide mechanisms for generating such raw strings.

`image = x.diff(input [,absolute=1])`	
	This method computes a new image that is the pixel by pixel (component by component) difference between image ‘x’ and image ‘input’. By default, the subtraction is performed as a signed operation which is clamped to [0,255] on output. If the ‘absolute’ keyword is set, the output is the absolute value of the difference clamped to the range [0,255].

`dict = x.stats()`	
	This method computes basic statistics on the image object. It computes for all pixels and for all channels (a 10x10 RGB image is considered to have 300 pixels). It returns a dictionary with the keys: ‘count’ - the number of pixels in the image ‘mean’ - the mean pixel value ‘sum’ - the sum of all the pixel values ‘sumofsquares’ - the sum of the squares of all of the pixels ‘variance’ - the variance of the pixels ‘minimum’- the minimum pixel value ‘maximum’ - the maximum pixel value

`x.bitblt(src,spos,ssize,pos,size,mode,backpixel)`

This method copies a rectangle of pixels from one image to the target. The two images must have the same pixel format. The rectangles may lie outside of the source or destination images. For source pixels that lie outside of the source image, the backpixel value is used. The source pixels will be scaled as needed (nearest neighbor sampling) to fill the destination rectangle. The operation mode can be a simple copy (`enve.CVF_BITBLT_COPY`), where source pixels replace destination pixels or it can be a masked copy (`enve.CVF_BITBLT_ALPHAMASK`) where source pixels only replace destination pixels if they have a non-zero alpha channel. backpixel is a tuple of 4 integers (Red,Green,Blue,Alpha). src is an enve.image() object to use as the pixel source. spos is the location of the pixels in the source image (a tuple of two ints). ssize is the rectangle (dx,dy) of the pixels in the source image (a tuple of two ints). pos is the location of the pixels in the destination image (a tuple of two ints). size is the rectangle (dx,dy) of the pixels to paint in the destination (a tuple of two ints).

`x.blend(src,srcCM,srcAM,dstCM,dstAM,fixedcolor,fixedalpha)`

This method blends the pixels of two images together according to four blending mode functions (a color and alpha for each of the source and destination images). Blending is performed by weighting each component of the source and destination by a particular function and then summing the results. The results are then clamped to the range [0,255] and stored in the destination image. In addition to the images, a fixed color and alpha value are supplied. These are used by some of the various expression combinations. fixedcolor is a tuple of four integers (RGBA) and fixedalpha is a single integer.

The formula take the following form:

```
dst'(RGB)=(dst(RGB)*dstCModeWeight)/255+(src(RGB)*srcCModeWeight)/255
dst'(A)  =(dst(A)  *dstAModeWeight)/255+(src(A)  *srcAModeWeight)/255
```

`srcCM` selects the function for determining `srcCModeWeight`
`srcAM` selects the function for determining `srcAModeWeight`
`dstCM` selects the function for determining `dstCModeWeight`
`dstAM` selects the function for determining `dstAModeWeight`

The available weight functions include:

```
enve.CVF_BLEND_ZERO       The value 0
enve.CVF_BLEND_ONE        The value 255
enve.CVF_BLEND_SCOLOR     The value of the source color
enve.CVF_BLEND_SALPHA     The value of the source alpha
enve.CVF_BLEND_DCOLOR     The value of the destination color
enve.CVF_BLEND_DALPHA     The value of the destination alpha
enve.CVF_BLEND_CCOLOR     The value of the "fixed" color
enve.CVF_BLEND_CALPHA     The value of the "fixed" alpha
enve.CVF_BLEND_MSCOLOR    255 - CVF_BLEND_SCOLOR
enve.CVF_BLEND_MSALPHA    255 - CVF_BLEND_SALPHA
enve.CVF_BLEND_MDCOLOR    255 - CVF_BLEND_DCOLOR
enve.CVF_BLEND_MDALPHA    255 - CVF_BLEND_DALPHA
enve.CVF_BLEND_MCCOLOR    255 - CVF_BLEND_CCOLOR
enve.CVF_BLEND_MCALPHA    255 - CVF_BLEND_CALPHA
```

err = x.resize(xsize,ysize[,smooth=sm])
This method resizes the current image to the size specified by xsize, ysize. This is essentially the same as changing the 'dim' attribute on the image object. By default, this operation is performed using nearest neighbor sampling. The caller may select higher quality box filtered sampling by setting the SMOOTH keyword equal to the value enve.CVF_SCALE_SMOOTH.

err = x.load(filename)
This method reads an image file and sets the current image object to its contents. If the input file is an animation, the first frame is returned. The method returns 0 on success or a negative integer on error. See enve.codecs() for supported file formats.

err = x.save(filename[,options=options_string])
This method saves the current image to disk in the specified filename. The format is selected by the filename extension. The target file is overwritten. The options keyword allows the user to set specific format options (e.g. compression) like the enve.movie options attribute. The method returns 0 on success or a negative integer on error. See enve.codecs() for supported filename extensions and option strings.

str = x.errstr()
If any of the methods return an error (-1), a string describing the error will be stored in the image object. These error strings can be accessed via this method.

print x
Prints basic information about the image x.

x == y
The object supports the comparison of two images for pixel by pixel equality

`x.attr` `x.attr = y`		
	These get and set the various image object attributes.	
	`__members__`	Returns a list of the attributes this object supports. Readonly List.
	`__methods__`	Returns a list of the methods this object supports. Readonly List.
	`dims`	Returns the size of the image in the x and y dimensions. If set, it will resize the image to the new x and y values using nearest neighbor sampling. Read/write tuple of ints (dx,dy).
	`framenumber`	Returns an index number that can be used to store a sequence number for this frame. Presently, this is unused. Read/write int.
	`frame`	Returns the size of the image frame for which this image might be a subrect of. Presently, this is unused. Read/write tuple of ints (dx,dy).
	origin	Returns the position (relative to the upper left corner) of the current image in the full image frame specified by the frame attribute. Presently, this is unused. Read/write tuple of ints (dx,dy).

	`format`	Returns the format of the image. If set, will change the image format to the new value (interpreting all pixel values).
		Valid values:
		`enve.CVF_IMG_FMT_A`
		`enve.CVF_IMG_FMT_L`
		`enve.CVF_IMG_FMT_LA`
		`enve.CVF_IMG_FMT_RGB`
		`enve.CVF_IMG_FMT_RGBA`
		`Read/write int.`

Additional EnVe API

`list=enve.codecs()`

This function queries the installed UDILs and returns a list of supported file formats and associated options. It returns a list structure in the following form:

```
[
   ["name","desc",[".ext",...],stereo,multi,
             [["optname",opttype,<optvalue>,<optdefault>],...]],
   ...
]
```

"`name`" is the name of the UDIL, this is used as the movie.format attribute.
"`desc`" is an ASCII description of the UDIL in more detail.

The next list is a list of the filename extensions used by this format.

`stereo` is an integer that is non-zero if the file supports HW stereo movies.
`multi` is an integer that is non-zero if the format is an animation file that places all frames in a single file.

The final list is a list of custom options that can be set for the format.

Each option has a name ("`optname`") and a type. The types can be:

```
enve.CVF_UDIL_BOOLEAN
enve.CVF_UDIL_INT
enve.CVF_UDIL_FLOAT
enve.CVF_UDIL_CHOICE.
```

<`optvalue`> and <`optdefault`> are different for each type:

Type	Value	Default
enve.CVF_UDIL_BOOLEAN	"0|1"	int
enve.CVF_UDIL_INT	[int_min,int_max]	int
enve.CVF_UDIL_FLOAT	[flt_min,flt_max]	float
enve.CVF_UDIL_CHOICE	["opt0","opt1",...]	int (index into list)

`["copyright", "ver",ver] = enve.version()`

This function returns a copyright notice and the version of the enve API as a text string and a floating point number.

6.4 EnSight extension mechanism

A number of features of EnSight are written as extensions in Python or command language. In EnSight, this system has been written to more fully integrate GUI objects and command language extensions into EnSight. The extension framework itself is written in Python and resides in $CEI_HOME/ensight100/site_preferences/extensions. This directory contains two directories: core and user_defined. The ‘core’ directory contains the Python implmentation of all of the base classes described here. The ‘user_defined’ directory contains the various EnSight extensions that are loaded on startup. At least two replacement GUI extensions, ten Tools and a large number of Menus are included by CEI in the distribution. The source code to these are an extremely valuable reference to the extension developer.

Startup process

At startup, EnSight builds a default sys.path which includes ‘$CEI_HOME/ensight100/site_preferences/extensions’ as well as `EnSightDefaultDirectory/extensions` (if it exists). EnSightDefaultsDirectory is %HOMEDRIVE%%HOMEPATH%\(username)\.ensight100 commonly located at C:\Users\username\.ensight100 on Vista and Win7, C:\Documents and Settings\yourusername\.ensight100 on older Windows, and ~/.ensight100 on Linux, and ~/Library/Application Support/ensight100 on the Mac. It will also add any directories specified by the CEI_PYTHONPATH environmental variable. This variable may contain multiple directory names, separated by a ‘:’ under unix operating systems and a ‘;’ under Windows.

EnSight scans in sys.path, looking for entries that end in the name ‘extension’. If any subdirectory of an extension directory contains an __init__.py file, the subdir is ‘import’ed into the module ensight.ext (note: ensight.ext.__path__ is modified to include the ‘extensions’ dirnames). If a user has common Python utilities or a startup script, it can be added as a module to the internal Python interpreter via this mechanism.

Formal EnSight extensions

EnSight supports the concept of a formal extension. An extension to EnSight is a subclass of one of the base-classes of the extension framework. Common extensions are for user-defined menus or “tools” and to replace the EnSight GUI itself. All extensions include a mechanism to extend EnSight command language to include custom batch scripting and action logging. EnSight looks for extensions by scanning all of the .define, .py and .enc files located in subdirectories rooted by a subdirectory of a sys.path ‘extension’ directory (as described above) named ‘user_defined’ for specially formatted comment blocks. There are two comment block forms for “Legacy” and “Direct load” extensions.

Legacy extensions

Legacy extensions can include operatons written in command language or Python. The comment block in each of these extensions has the general form:

```
#ENSIGHT_USER_DEFINED_BEGIN
#TEXT=name used for GUIs
#TOOLTIP=text displayed in tooltips
#ICON=name_of_an_iconfile.png
#NAME=simplename
#DESC=text description of the item
#TYPE=TOOL,MENU,TOOLDIR,MENUDIR
#MODE='All','Part','Annot','Plot','VPort','Frame'
#PTYPE='All','None','Mixed','Model','ClipPlane','Contour','DiscretePar
ticle','Frame','IsoSurface','ParticleTrace','Profile','VectorArrow','E
```

```
levatedSurface','DevelopedSurface','ModelExtract','ModelCut','ModelBou
ndary','IsoVolume','BuiltUp','TensorGlyph','FxVortexCore','FxShock','F
xSepAtt','MaterialInterface','Point','AxiSymmetric','ModelMerge','Mult
'
#ENSIGHT_USER_DEFINED_END
```

Most of these fields are optional. If they are included in a .py or .enc file, they inform EnSight that these are TOOL or MENU extensions (depending on the TYPE= value). EnSight will build an extension object (a subclass of the tool_extension or menu_extension classes) for the file and if the extension is executed, the contents of the file will be executed. When it is run, the directory containing the file will be added to sys.path temporarily.

MODE= and PTYPE= fields are used for filtering the display of the item depending on the current state of EnSight (what mode and what type of parts are selected).

A file named ‘menu.define’ or ‘tool.define’ can contain just this comment block, but uses the type ‘TOOLDIR’ or ‘MENUDIR’. If this file exists, the other commented files are included as GUI children of that object. The loader will instantiate a class of the proper type: ‘menu_extension’ or ‘tool_extension’ and registers it with EnSight. The registration operation has two parts. First, the class is stored under the proper key in the dictionary ‘ensight.core.extensions’ so that object can query and use the various extensions. Second, the register method is called on the base class (’core_extension’). This methods registers a method on the object as a the command language extension. Thus, an object with the name ‘foo’ that is a child of an object with the name ‘bar’ will be able to generate command language of the form “ext: bar.foo string”. It is also able to handle the playback of such strings by overriding the base class.

Direct loaded (Python) extensions

Direct load extensions can only be written in Python and have a number of advantages over Legacy extiensions, including private namespaces in both Python and command language. Developers are encouraged to use this form of extension whenever possible as it avoid potential namespace collisions both in Python and EnSight. To use this mechanism, the developer writes their own subclass(es) of the appropriate extension base class(es). They then add a comment block to the top of the Python source code file that looks like this:

```
#ENSIGHT_USER_DEFINED_BEGIN
#FACTORY=function_name
#ENSIGHT_USER_DEFINED_END
```

In this case, the file will be imported and the function ‘function_name()’ will be invoked with a single ‘parent’ parameter. The function should create all of the instances of the extension class objects needed and return them as a Python list. Note that function_name() should also do any parent adding for the objects it wants before returning them. The calling system will register the objects in the returned list and insert them into the ensight.core.extension dictionary. When this file is imported, the directory it is in will have been added to the sys.path variable.

Extension base classes

The extension mechanism is based on Python subclasses. All extensions subclass from a code base class. This class provides two important features. First, each extension is in a private namespace in Python, allowing each one to have its own “global” variables. Second, each extension is given part of the command language namespace complete with methods to generate custom command language and replay it. The simplest extension just extends command language with a new command (note: in command language, all extensions appear as: ‘ext: namespace string’). The base classes are “core’, “guibase”, “gui”, “tool” and “menu”.

core_extension class

Basic extension of the command language (type = “core”). Override the cmdExec() and cmdRecord() methods to generate your own command language extensions.

Data members:

_version : version number
_path : path to the file defining this extension
_name : “name” for purposes of Python/cmdlang namespaces (no spaces, odd chars, etc)
_text : Display name of the extension (spaces allowed)
_desc : ASCII description of the exension (spaces allowed)
_namespace : actual cmdlang namespace (e.g. ext: namespace foo)
_children : list of children of this extension

Member functions:

addChild(child) : add a child
setDesc(text)
setSort(text) : set the menu sorting value for this extension
cmdExec(string) : called back when cmdlang sees “ext: namespace foo” String will be “foo”
cmdRecord(string) : call to record command language “ext: _namespace string”

guibase_extension class (subclass of core_extension)

This class adds common GUI elements such as itcons and tooltips (type = “guibase”). In general, user defined extensions should not subclass this class directly.

Member functions:

setInMenus(v) : Set the state of dynamic menu visiblity
getInMenus() : True if this extension should show up in any dynamic menus
setTooltip(text) : the text for the tooltip
setIcon(”pathname”) : filename for the icon, can be a resource reference (e.g. “:/ensight/...”)
getWidget(w,parent) : return the widget representation of the object. This method is overridden by subclasses
setWaitCursor(onoff) : display/clear the hourglass cursor for this extension

gui_extension class (subclass of guibase_extension)

A full user-defined GUI (type = “gui”). Subclass this class to completely replace the EnSight GUI with your own gui. Note: the cmdExec method has been overridden to include “activate_gui”

Member functions:

setSplash(pathname) : pathname to an SVG file to use as the splash screen
doActivate(onoff) : called to enable/disable this GUI must be overridden, returns False on failure.
doErrorMessage(message,style) : called when EnSight throws an error or needs a Yes/No question, must be overridden.
doGuiEvent(message,targetlist) : called when there is a GUI event, must be overridden.

Useful module functions:

These functions can be used directly by applications that want to interact with the core GUI and its events.

ensight.core.gui_extension.gui_activate(name)
gui=ensight.core.gui_extension.current_active_gui()
ensight.core.gui_extension.error_message(msg,what)
ensight.core.gui_extension.gui_event(msg,targets)

tool_extension class (subclass of guibase_extension)

Baseclass for an EnSight user-defined tool (type = “tool”). All tool extensions must subclass from this class.

Member functions:

run() : This function is called when the tool is invoked. The function must be overridden.

menu_extension class (subclass of guibase_extension)

Baseclass for an EnSight popup menu (type = “menu”). All menu extensions must subclass from this class.

Data members:

_info : the dictionary of values resulting from the mousedown that triggered this menu

Member functions:

setSeparator(boolean) : set to true if this menu is a separator
setMode(mode) : a string containing the mode names for this menu to be displayed in. Concatenated from: ‘Part’,’Annot’,’Plot’,’VPort’,’Frame’,’All’
setPartType(type) : a string containing the part types for which this menu is to be displayed. Concatenated from:
‘All’,’Model’,’ClipPlane’,’Contour’,’DiscreteParticle’,’Frame’,’IsoSurface’,’ParticleTrace’,’Profile’,’VectorArrow’,’ElevatedSurface’,’DevelopedSurface’,’ModelExtract’,’ModelCut’,’ModelBoundary’,’IsoVolume’,’BuiltUp’,’TensorGlyph’,’FxVortexCore’,’FxShock’,’FxSepAtt’,’MaterialInterface’,’Point’,’AxiSymmetric’,’ ModelMerge’,’Mult’
validFilter(self,emode,ptype) : return 1 if this menu should be shown. By default, it filters using the mode and parttype values as noted above, however for general filtering, one ma override this method.
run() : override this method do what this menu should do when selected.

the _info attribute is set to the “context” of the operation that started the menu operation. It is a dictionary. Some fields are pre-defined, depending on the EnSight RMB context.

The pre-defined _info fields are:

Common to all modes

‘pick’: the click target

‘pos’: the (x,y,z) tuple of the click in data space

‘target’: list of ensight object references that are the action target (list can be empty)

‘mousepos’: (x,y) position tuple in the graphics window of the mouse down (normalized coords). -1,-1 if there is no position or one is not in the window.

pick: CVF_CURSOR_TOOL

pick: CVF_LINE_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_END2, CVF_EVENT_LOCAT_END1, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS

pick: CVF_PLANE_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_UPPER_RIGHT, CVF_EVENT_LOCAT_LOWER_LEFT, CVF_EVENT_LOCAT_LOWER_RIGHT, CVF_EVENT_LOCAT_UPPER_LEFT, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS

pick: CVF_CYLINDER_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_END2, CVF_EVENT_LOCAT_END1, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS, CVF_EVENT_LOCAT_RADIUS

pick: CVF_CONE_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_END2, CVF_EVENT_LOCAT_END1, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS, CVF_EVENT_LOCAT_RADIUS

pick: CVF_SPHERE_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_END2, CVF_EVENT_LOCAT_END1, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS

pick: CVF_REVO_TOOL

'item': The part of the line clicked: CVF_EVENT_LOCAT_MID, CVF_EVENT_LOCAT_END2, CVF_EVENT_LOCAT_END1, CVF_EVENT_LOCAT_XAXIS, CVF_EVENT_LOCAT_YAXIS, CVF_EVENT_LOCAT_ZAXIS

pick: CVF_POLYLINE

'item': The spline number clicked on

'point': The index into the spline knot points

pick: CVF_LEGEND

'item': The palette id

'component': The palette id component (for vectors)

pick: CVF_ANNOT

'item': The annotation id (the target object has the type implicitly)

pick: CVF_PART

'item': The part id

pick: CVF_VIEWPORT

'item': The viewport id

pick: CVF_PLOTTER

'curve': The curve number clicked on

'value': The (f,x) value at the clicked point

'_prepick_': This key is optional. If set to the name of a user-defined menu, the doPopup() function will not display any menus. Instead, it will call the 'run()' method on the first menu with that name. This mechanism makes it easy to execute a menu in a given spatial context.

Useful module functions:

core.userdefinedmenus.doPopup(infodict) : popup the current RMB menu at the mouse. You must pass a proper info dictionary (see: _info above)

With this function, the caller needs to create the _info dictionary. One special key has been defined: 'mode'. If this key is present, it will override the current EnSight mode value for validFilter() filtering. An example using this function would be:

```
dict = {'pick': ensight.CVF_PART, 'item': partid, 'target': partobj,
        'pos': (0,0,0), 'mousepos': (-1,-1) }
ensight.core.userdefinedmenus.doPopup(dict)
```

Internationalization and EnSight Extensions

The EnSight core includes an instance of the CEItranslate object. It can be accessed as: ensight.core.tr. When the ensight.core module loads and scans the directories for extensions, it also scans for any *.qm files and automatically registers them with ensight.core.tr. Thus, if you have internationalized an extension GUI (or any other Qt/PyQt gui), simply place the translation files in the extensions directory and they will be loaded. When the QApplication instance is created, ensight.core.tr.changelang() is called. If the user has set CEI_LANG (or the host system has a language set), the translation files for that langauge are automatically loaded. CEI_LANG should be set to the two character ISO 639 code for the target language.

In Python code, you can trap and handle dynamic language changes by including the following method to your top level widget:

```
# i18n bits: handle dynamic language changes
    def changeEvent(self, event):
        if (event.type() == QtCore.QEvent.LanguageChange):
            self.retranslateUi(self)
        QtGui.QMainWindow.changeEvent(self, event)
```

Note that you may need to change 'QMainWindow' to the appropriate classname. The 'self.retranslateUi()' call should be replaced with whatever function regenerates the GUI text contents on your system (it may be more than one call to autogenerated retranslateUi() methods.

6.5 Helper modules

CEI has included a number of helper modules that wrapper common functions. These are included in a couple of different modules. Python developers are encouraged to familiarize themselves with these modules and their functions as they can save a great deal of development time and provide a more consistent user experience.

'ensight.core' module

This module include a number of image saving and printing functions that implement standard GUI menus.

CeiEnSightPrintWindow() class for rasterizing and printing (in qtimageutils.py)

CeiEnSightCopyWindow() copy to clipboard (in qtimageutils.py)

CeiEnSightSaveImage() save image to disk (in qtimageutils.py)

CeiEnSightImportImageAnnot() import image file/movie as an annotation (in qtimageutils.py)

In the core, there are some useful items (created by __init__.py). ensight.core.ENS_stdout_instance/ ENS_stderr_instance are the wrappers for the Python interpreter output. EnSight has 'resume()' and 'suspend()' methods which allows output to be redirected to the console instead of the command dialog Python tab. There is also a dictionary 'ensight.core.extensions'. The keys of the dict include 'menu', 'tool', 'core', 'gui' and each points to a list of instances of object of the class 'xxxx_extension's that were created on startup.

'cei' module

This module includes a number of common functions that are not generally EnSight specific. They are used in tools like EnVe, but can be used by any EnSight Python application.

CeiPrintDlg() class for PyQt based rendering and printing of the main window (in printdlg.py)

CeiQtImageFormatOptions() class for prompting the user for UDIL selection/options (in qtimgformats.py)

CeiQtSaveImage() class to prompt the user for a complete set of image size/saving options, including the render and save) (in qtimgformats.py)

CeiPlot() classes to wrapper matplotlib. Create a graph and save it to disk (in plots.py)

CeiQtPlotWindow() class providing a dockable Qt4 widget that contains a CeiPlot object, including copy and print operations (in qtplots.py)

CEItranslate() class that acts as a repository for .qm translation files and allows for dynamic changing of the language for all objects as well as a method to get a language menu (in qtlanguage.py)

ceisplash() class to manage a splash screen from a .svg file (in ceisplash.py)

CeiLogging() and logging shortcuts (in ceilogging.py)

CeiQtGenericDialog() class that generates dynamic dialogs from simple lists and prompts the user for values. It can handle things like text, checkboxes, ints, floats, variables, parts, option menus, strings, filenames, etc (in qtgenericdlg.py)